OFFICE COPY
4/73
DEC-11-ASDD-D
DIGITGRAPH
P. O. Box 4240
Tucson, Arizona 85

PAL-11R Assembler, Edit-11 Text Editor

ODT-11R Debugging Program, PIP File Utility Package,

Link-11 Linker and Libr-11 Librarian

programmer's manual

P A L - 1 1 R A S S E M B L E R

P R O G R A M M E R ' S M A N U A L

Program Assembly Language

and

Relocatable Assembler

for the

Disk Operating System

September 1971

SOFTWARE SUPPORT CATEGORY

The software described in this document is supported by DEC under category I, as defined on page iv of this document.

For additional copies, order No. DEC-11-ASDD-D, from Digital Equipment Corporation, Direct Mail, Building 1-1, Maynard, Mass. 01754

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

First Printing, February 1971
Revised May 1971
Revised September 1971
Revised April 1972

Your attention is invited to the last two pages of this document. The "How to Obtain Software Information" page tells you how to keep up-to-date with DEC's software. The "Reader's Comments" page, when filled in and mailed, is beneficial to both you and DEC; any comments received are acknowledged and are considered when documenting subsequent manuals.

Copyright © 1971, 1972 by Digital Equipment Corporation

This document is for information purposes only, and is subject to change without notice.

Associated Documents:

PDP-11 Disk Operating System Monitor,
Programmer's Handbook, DEC-11-MWDC-D

PDP-11 Edit-11 Text Editor,
Programmer's Manual, DEC-11-EEDA-D

PDP-11 ODT-11R Debugging Program,
Programmer's Manual, DEC-11-OODA-D

PDP-11 PIP, File Utility Package,
Programmer's Manual, DEC-11-PIDB-D

PDP-11 Link-11 Linker and Libr-11 Librarian,
Programmer's Manual, DEC-11-SLDC-D

PDP-11 FORTRAN IV,
Programmer's Manual, DEC-11-KFDA-D

The following are trademarks of Digital Equipment Corporation:

DEC	PDP
FLIP CHIP	FOCAL
DIGITAL (logo)	COMPUTER LAB
UNIBUS	OMNIBUS

P R E F A C E

This manual describes PAL-11R, the Assembly Language and Assembler for the PDP-11 Disk Operating System. Thus, familiarity with the DOS Monitor (see PDP-11 Disk Operating System Monitor, Programmer's Handbook, DEC-11-MWDC-D) is assumed.

The manual explains how to write PAL-11R source programs and how to assemble them into object modules. All facets of the assembly language are explained and illustrated with many examples, and the manual concludes with assembling procedures.

In explaining how to write PAL-11R source programs, it is necessary, especially at the outset, to make frequent forward references. Therefore, we recommend that you first read through the entire manual to get a "feel" for the language, and then reread the manual for a complete understanding of the language and assembling procedures.

In addition to the PAL-11R Assembler, the Disk Operating System software includes:

DOS Monitor
Edit-11 Text Editor
ODT-11R Debugging Program
PIP, File Utility Package
Link-11 Linker
Libr-11 Librarian
FORTRAN IV

SOFTWARE SUPPORT CATEGORIES

Digital Equipment Corporation (DEC) makes available four categories of software. These categories reflect the types of support a customer may expect from DEC for a specified software product. DEC reserves the right to change the category of a software product at any time. The four categories are as follows:

CATEGORY I
Software Products Supported at no Charge

This classification includes current versions of monitors, programming languages, and support programs provided by DEC. DEC will provide installation (when applicable), advisory, and remedial support at no charge. These services are limited to original purchasers of DEC computer systems who have the requisite DEC equipment and software products.

At the option of DEC, a software product may be recategorized from Category I to Category II for a particular customer if the software product has been modified by the customer or a third party.

CATEGORY II
Software Products that Receive Support for a Fee

This category includes prior versions of Category I programs and all other programs available from DEC for which support is given. Programming assistance (additional support), as available, will be provided on these DEC programs and non-DEC programs when used in conjunction with these DEC programs and equipment supplied by DEC.

CATEGORY III
Pre-Release Software

DEC may elect to release certain software products to customers in order to facilitate final testing and/or customer familiarization. In this event, DEC will limit the use of such pre-release software to internal, non-competitive applications. Category III software is only supported by DEC where this support is consistent with evaluation of the software product. While DEC will be grateful for the reporting of any criticism and suggestions pertaining to a pre-release, there exists no commitment to respond to these reports.

CATEGORY IV
Non-Supported Software

This category includes all programs for which no support is given.

C O N T E N T S

			Page
CHAPTER 1		INTRODUCTION	1-1
CHAPTER 2		STATEMENTS	2-1
	2.1	CHARACTER SET	2-1
	2.2	LABEL	2-2
	2.3	OPERATOR	2-3
	2.4	OPERAND	2-3
	2.5	COMMENTS	2-4
	2.6	FORMAT CONTROL	2-4
CHAPTER 3		SYMBOLS	3-1
	3.1	PERMANENT SYMBOLS	3-1
	3.2	USER-DEFINED SYMBOLS	3-1
	3.3	DIRECT ASSIGNMENT	3-2
	3.4	REGISTER SYMBOLS	3-3
CHAPTER 4		EXPRESSIONS	4-1
	4.1	NUMBERS	4-1
	4.2	ARITHMETIC AND LOGICAL OPERATORS	4-2
	4.3	ASCII CONVERSION	4-2
	4.4	MODE OF EXPRESSIONS	4-3
CHAPTER 5		ASSEMBLY LOCATION COUNTER	5-1
CHAPTER 6		RELOCATION AND LINKING	6-1
CHAPTER 7		ADDRESSING	7-1
	7.1	REGISTER MODE	7-2
	7.2	DEFERRED REGISTER MODE	7-2
	7.3	AUTOINCREMENT MODE	7-2
	7.4	DEFERRED AUTOINCREMENT MODE	7-3
	7.5	AUTODECREMENT MODE	7-3
	7.6	DEFERRED AUTODECREMENT MODE	7-3
	7.7	INDEX MODE	7-4

		Page
7.8	DEFERRED INDEX MODE	7-4
7.9	IMMEDIATE MODE AND DEFERRED IMMEDIATE (ABSOLUTE) MODE	7-4
7.10	RELATIVE AND DEFERRED RELATIVE MODES	7-5
7.11	TABLE OF MODE FORMS AND CODES	7-6
7.12	INSTRUCTION FORMS	7-7
CHAPTER 8	ASSEMBLER DIRECTIVES	8-1
8.1	.TITLE	8-1
8.2	.GLOBL	8-1
8.3	PROGRAM SECTION DIRECTIVES	8-2
8.4	.EOT	8-5
8.5	.EVEN	8-5
8.6	.END	8-5
8.7	.WORD	8-5
8.8	.BYTE	8-7
8.9	.ASCII	8-7
8.10	.RAD50	8-8
8.11	.LIMIT	8-9
8.12	CONDITIONAL ASSEMBLY DIRECTIVES	8-9
CHAPTER 9	OPERATING PROCEDURES	9-1
9.1	LOADING PAL-11R	9-1
9.2	INITIAL DIALOGUE	9-1
9.3	ASSEMBLY DIALOGUE	9-6
9.4	ASSEMBLY LISTING	9-7
9.5	OBJECT MODULE OUTPUT	9-8
9.5.1	Global Symbol Directory	9-8
9.5.2	Text Block	9-9
9.5.3	Relocation Directory	9-9
CHAPTER 10	ERROR CODES	10-1
APPENDIX A	ASCII CHARACTER SET	A-1
APPENDIX B	PAL-11R ASSEMBLY LANGUAGE AND ASSEMBLER	B-1
B.1	TERMINATORS	B-1
B.2	ADDRESS MODE SYNTAX	B-2
B.3	INSTRUCTIONS	B-3

		Page
B.3.1	Double Operand Instructions	B-4
B.3.2	Single Operand Instructions	B-4
B.3.3	Rotate/Shift	B-5
B.3.4	Operate Instructions	B-6
B.3.5	Trap Instructions	B-7
B.3.6	Branch Instructions	B-7
B.3.7	Subroutine Call	B-8
B.3.8	Subroutine Return	B-8
B.4	ASSEMBLER DIRECTIVES	B-8
B.5	ERROR CODES	B-9

APPENDIX C	LINKING PAL-11R AND ITS OVERLAY BUILDER, AND CONSTRUCTING THE RUN TIME SYSTEM	C-1
APPENDIX D	PERMANENT SYMBOL TABLE	D-1
APPENDIX E	SYSTEM ERROR MESSAGES	E-1
APPENDIX F	VØØ5A PROGRAMMING NOTES	F-1
APPENDIX G	FEATURES OF VØØ7A	G-1

CHAPTER 1

INTRODUCTION

PAL-11R (Program Assembly Language for the PDP-11, Relocatable Version) operates under the Disk Operating System (DOS). PAL-11R enables you to write source (symbolic) programs using letters, numbers, and symbols which are meaningful to you. The source programs, generated either on-line using the Text Editor, Edit-11 (see PDP-11 Edit-11 Programmer's Manual, DEC-11-EEDA-D), or off-line; are then assembled into object modules which are processed by the PDP-11 Linker, Link-11. Link-11 produces a load module which is loaded by the monitor command RUN for execution. Object modules may contain absolute and/or relocatable code, and separately assembled object modules may be linked with global symbols. The object module is produced after two passes through the Assembler. A complete octal/symbolic listing of the assembled program may also be obtained. This listing is especially useful for documentation and debugging purposes.

Some notable features of PAL-11R are:

1. Selective assembly pass functions
2. Device and file name specification for pass functions
3. Error listing on command output device
4. Double buffered and concurrent I/O
5. Alphabetized, formatted symbol table listing
6. Relocatable object modules
7. Global symbols for linking between object modules
8. Conditional assembly directives
9. Program sectioning directives

The following discussion of the PAL-11R Assembly Language assumes that you have read the PDP-11 Handbook 1971, with emphasis on those sections which deal with the PDP-11 instruction set, formats, and timings -- a thorough knowledge of these is vital to efficient assembly language programming.

CHAPTER 2

STATEMENTS

A source program is composed of a sequence of statements, where each statement is on a single line. The statement is terminated by a carriage return character and must be immediately followed by either a line feed or form feed character. Should a carriage return character be present and not be followed by a line feed or form feed, the Assembler will generate a Q error (Chapter 10), and that portion of the line following the carriage return will be ignored. Since the carriage return is a required statement terminator, a line feed or form feed not immediately preceded by a carriage return will have one inserted by the Assembler.

It should be noted that, if the Edit-11 Text Editor is being used to create the source program, a typed carriage return (RETURN key) automatically generates a line feed character.

A statement may be composed of up to four fields which are identified by their order of appearance and by specified terminating characters as explained below and summarized in Appendix B. The four fields are:

Label Operator Operand Comment

The label and comment fields are optional. The operator and operand fields are interdependent, i.e., either may be omitted depending upon the contents of the other.

2.1 CHARACTER SET

A PAL-11R source program is composed of symbols, numbers, expressions, symbolic instructions, assembler directives, argument separators, and line terminators written using the following ASCII[1] characters.

1. The letters A through Z. (Upper and lower case letters are acceptable, although upon input, lower case letters will be converted to upper case letters.)

[1]ASCII stands for American Standard Code for Information Interchange.

2. The numbers 0 through 9.
3. The characters . and $ (these characters are reserved for systems use).
4. The separating or terminating symbols:

   : = % # @ ( ) , ; " ' + - & !

   carriage return tab space line feed form feed

2.2 LABEL

A label is a user-defined symbol (Chapter 3) which is assigned the value of the current location counter. This value may be either absolute or relocatable, depending on whether the location counter value is absolute or relocatable. In the latter case, the final absolute value is assigned by the Linker, i.e., the value plus the relocation constant. A label is a symbolic means of referring to a specific location within a program. If present, a label always occurs first in a statement and must be terminated by a colon. For example, if the current location is absolute 100_8, the statement:

```
ABCD:     MOV A,B
```

will assign the value 100_8 to the label ABCD so that subsequent reference to ABCD will be to location 100_8. In the above example if the location counter were relocatable, the final value of ABCD would be 100_8+K, where K is the location of the beginning of the relocatable section in which the label ABCD appears. More than one label may appear within a single label field; each label within the field will have the same value. For example, if the current location counter is 100, multiple labels in the statement:

```
ABC:     $DD:     A7.7:     MOV A,B
```

will equate each of the three labels ABC, $DD, and A7.7 with the value 100_8. ($ and . are reserved for system software.)

The error code M (multiple definition of a symbol) will be generated during assembly if two or more labels have the same first six characters.

2.3 OPERATOR

An operator follows the label field in a statement, and may be an instruction mnemonic or an assembler directive (Appendix B). When it is an instruction mnemonic, it specifies what action is to be performed on any operand(s) which follows it. When it is an assembler directive, it specifies a certain function or action to be performed during assembly.

The operator may be preceded only by one or more labels and may be followed by one or more operands and/or a comment. An operator is legally terminated by a space, tab, or any of the following characters:

```
#  +  -  @  (  "  '  %  !  &  ,  ;
line feed      form feed    carriage return
```

The use of each character above will be explained.

Consider the following examples:

```
MOV→|A,B        ; →| (TAB) terminates operator MOV
MOV@A,B         ; @ terminates operator MOV
```

When the operator stands alone without an operand or comment, it is terminated by a carriage return followed by a line feed or form feed character.

2.4 OPERAND

An operand is that part of a statement which is operated on by the operator -- an instruction mnemonic or assembler directive. Operands may be symbols, expressions, or numbers. When multiple operands appear within a statement, each is separated from the next by a comma. An operand may be preceded by an operator and/or label, and followed by a comment.

The operand field is terminated by a semicolon when followed by a comment, or by a carriage return followed by a line feed or form feed character when the operand ends the statement. For example:

```
LABEL:  MOV GEORGE,BOB     ;THIS IS A COMMENT
```

where the space between MOV and GEORGE terminated the operator field and began the operand field; the comma separated the operands GEORGE and BOB; the semicolon terminated the operand field and began the comment.

2.5 COMMENTS

The comment field is optional and may contain any ASCII character except null, rubout, carriage return, line feed, or form feed. All other characters, even those with special significance, are ignored by the Assembler when used in the comment field.

The comment field may be preceded by none, any, or all of the other three fields. It must begin with the semicolon and end with a carriage return followed by a line feed or form feed character. For example:

```
LABEL:    CLR HERE          ;THIS IS A $1.00 COMMENT
```

Comments do not affect assembly processing or program execution, but they are useful in program listings for later analysis, checkout, or documentation purposes.

2.6 FORMAT CONTROL

Formatting of the source program is controlled by the space and tab characters. They have no effect on the assembling process of the source program unless they are embedded within a symbol, number, or ASCII text; or are used as the operator field terminator. Thus, they can be used to provide a neat, readable program. A statement can be written:

```
LABEL:MOV(SP)+,TAG;POP VALUE OFF STACK
```

or, using formatting characters, it can be written:

```
LABEL:    MOV (SP)+,TAG    ;POP VALUE OFF STACK
```

which is much easier to read.

Page size is controlled by the form feed character. A page of n lines is created by inserting a form feed (CTRL/FORM keys on the keyboard) after the nth line. If no form feed is present, a page is terminated after 56 lines.

CHAPTER 3

SYMBOLS

There are two types of symbols: permanent and user-defined. There are two symbol tables: the Permanent Symbol Table (PST) and the User Symbol Table (UST). The PST contains all the permanent symbols and is part of the Assembler's load module. The UST is constructed as the source program is assembled; user-defined symbols are added to the table as they are encountered.

3.1 PERMANENT SYMBOLS

Permanent symbols consist of the instruction mnemonics (Appendix B.3) and assembler directives (Chapter 8). These symbols are a permanent part of the Assembler and need not be defined before being used in the source program.

3.2 USER-DEFINED SYMBOLS

User-defined symbols are those defined as labels (Section 2.2) or by direct assignment (Section 3.3). These symbols are added to the User Symbol Table as they are encountered during the first pass of the assembly. They can be composed of alphanumeric characters, dollar signs, and periods only; any other character is illegal and, if used, will result in the error message I (Chapter 10). (Again, $ and . are reserved for system software.) The following rules also apply to user-defined symbols:

1. The first character must not be a number.
2. Each symbol must be unique within the first six characters.
3. A symbol may be written with more than six legal characters but the seventh and subsequent characters are only checked for legality, and are not otherwise recognized by the Assembler.
4. Spaces and tabs must not be embedded within a symbol.

A user-defined symbol may duplicate a permanent symbol. The value associated with a permanent symbol that is also user-defined depends upon its use:

1. A permanent symbol encountered in the operator field is

associated with its corresponding machine op-code.

2. If a permanent symbol in the operand field is also user-defined, its user-defined value is associated with the symbol. If the symbol is not found to be user-defined, then the corresponding machine op-code absolute value is associated with the symbol.

User-defined symbols are either internal or global. All symbols are internal unless they are explicitly defined as global with the .GLOBL assembler directive (Section 8.2).

Global symbols are used to provide links between object modules. A global symbol, which is defined (as a label or by direct assignment) in a program, is called an entry symbol or entry point. Such symbols may be referenced from other object modules or assemblies. A global symbol which is not defined in the current assembly is called an external symbol. Some other assembly must define the same symbol as an entry point.

Since PAL-11R provides program sectioning capabilities (Section 8.3), two types of internal symbols must be considered:

1. a symbol that belongs to the current program section;
2. a symbol that belongs to any of the other program sections.

In both cases, the symbol must be defined within the current assembly; the significance of the distinction is critical in evaluating expressions involving type 2. above (Section 4.4).

3.3 DIRECT ASSIGNMENT

A direct assignment statement associates a symbol with a value. When a direct assignment statement defines a symbol for the first time, that symbol is entered into the Assembler's symbol table and the specified value is associated with it. A symbol may be redefined by assigning a new value to a previously defined symbol. The newly assigned value will replace the previous value assigned to the symbol.

The symbol takes on the relocatable or absolute attribute of the defining expression. However, if the defining expression is global, the defined symbol will not be global unless previously defined as such (Chapter 4).

The general format for a direct assignment statement is:

```
symbol = expression
```

The following conventions apply:

1. An equal sign (=) must separate the symbol from the expression defining the symbol.
2. A direct assignment statement may be preceded by a label and may be followed by a comment.
3. Only one symbol can be defined by any one direct assignment statement.
4. Only one level of forward referencing is allowed.

Example of two levels of forward referencing (illegal):

```
X = Y
Y = Z
Z = 1
```

X and Y are both undefined throughout pass 1 and will be listed on the command output device as such at the end of that pass. X is undefined throughout pass 2, and will cause a U error message.

Examples:

```
        A = 1               ;THE SYMBOL A IS EQUATED WITH THE VALUE 1.
        B = 'A-1&MASKLOW    ;THE SYMBOL B IS EQUATED WITH THE EXPRES-
                            ;SION'S VALUE
C:      D = 3               ;THE SYMBOL D IS EQUATED WITH 3.  THE
E:      MOV #1,ABLE         ;LABELS C AND E ARE EQUATED WITH THE
                            ;NUMERICAL MEMORY ADDRESS OF THE MOV
                            ;COMMAND.
```

3.4 REGISTER SYMBOLS

The eight general registers of the PDP-11 are numbered 0 through 7. These registers may be referenced by use of a register symbol; that is, a symbolic name for a register. A register symbol is defined by means of a direct assignment, where the defining expression contains at least one term preceded by a % or at least one term previously defined as a register symbol. In addition, the defining

expression of a register symbol must be absolute. For example:

```
RØ=%Ø           ;DEFINE RØ AS REGISTER Ø
R3=RØ+3         ;DEFINE R3 AS REGISTER 3
R4=1+%3         ;DEFINE R4 AS REGISTER 4
THERE=%2        ;DEFINE THERE AS REGISTER 2
```

It is important to note that all register symbols must be defined before they are referenced. A forward reference to a register symbol will generally cause phase errors (Chapter 10).

The % may be used in any expression thereby indicating a reference to a register. Such an expression is a register expression. Thus, the statement:

```
CLR  %6
```

will clear register 6 while the statement:

```
CLR  6
```

will clear the word at memory address 6. In certain cases a register can be referenced without the use of a register symbol or register expression. These cases are recognized through the context of the statement and are thoroughly explained in Sections 7.11 and 7.12. Two obvious examples of this are:

```
JSR    5,SUBR    ;THE FIRST OPERAND FIELD MUST ALWAYS BE
                 ;A REGISTER.

CLR     X(2)     ;ANY EXPRESSION ENCLOSED IN ( ) MUST BE
                 ;A REGISTER.  IN THIS CASE, INDEX REGIS-
                 ;TER 2.
```

CHAPTER 4

EXPRESSIONS

Arithmetic and logical operators (see Section 4.2) may be used to form expressions. A term of an expression may be a permanent symbol, a user-defined symbol (which may be absolute, relocatable, or global), a number, ASCII data, or the present value of the assembly location counter represented by the period. Expressions are evaluated from left to right. Parenthetical grouping is not allowed.

Expressions are evaluated as word quantities. The operands of a .BYTE directive (Section 8.8) are evaluated as word expressions before truncation to the low-order eight bits. The evaluation of an expression includes the evaluation of the mode of the resultant expression; that is, absolute, relocatable or external. The definition of the modes of expression are given below in Section 4.4.

A missing term, expression or external symbol will be interpreted as 0. A missing operator will be interpreted as +. The error code Q (Questionable syntax) will be generated for a missing operator. For example:

```
A + -100                ;OPERAND MISSING
```

will be evaluated as A + 0 - 100, and:

```
TAG ! LA 177777         ;OPERATOR MISSING
```

will be evaluated as TAG ! LA+177777.

The value of an external expression will be the value of the absolute part of the expression; e.g., EXT+A will have a value of A. This will be modified by the Linker to become EXT+A.

4.1 NUMBERS

The Assembler accepts both octal and decimal numbers. Octal numbers consist of the digits 0 through 7 only. Decimal numbers consist of the digits 0 through 9 followed by a decimal point. If a number

contains an 8 or 9 and is not followed by a decimal point, the N error code (Chapter 10) will be printed and the number will be interpreted as decimal. Negative numbers may be expressed as a number preceded by a minus sign rather than in a two's complement form. Positive numbers may be preceded by a plus sign although this is not required.

If a number is too large to fit into 16 bits, the number is truncated from the left. In the assembly listing the statement will be flagged with a Truncation (T) error. Numbers are always considered to be absolute quantities (that is, not relocatable).

4.2 ARITHMETIC AND LOGICAL OPERATORS

The arithmetic operators are:

- `+` indicates addition or a positive number
- `-` indicates subtraction or a negative number

The logical operators are:

- `&` indicates the logical AND operation
- `!` indicates the logical inclusive OR operation

AND	OR
Ø & Ø = Ø	Ø ! Ø = Ø
Ø & 1 = Ø	Ø ! 1 = 1
1 & Ø = Ø	1 ! Ø = 1
1 & 1 = 1	1 ! 1 = 1

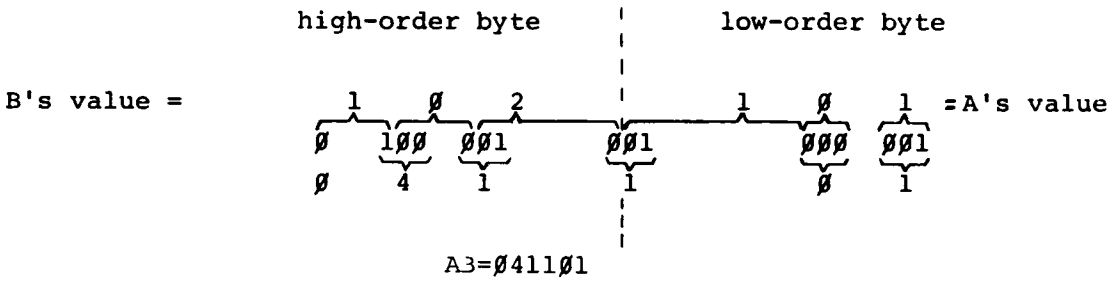
4.3 ASCII CONVERSION

When preceded by an apostrophe, any ASCII character (except null, rubout, carriage return, line feed, or form feed) is assigned the 7-bit ASCII value of the character (Appendix A). For example:

```
'A
```

is assigned the value 101_8.

When preceded by a quotation mark, two ASCII characters (not including null, rubout, carriage return, line feed, or form feed) are assigned the 7-bit ASCII values of each of the characters to be used. Each 7-bit value is stored in an 8-bit byte and the bytes are combined to form a word. For example "AB will store the ASCII value of A in the low-order (even) byte and the value of B in the high-order (odd) byte:



ASCII text is always absolute.

4.4 MODE OF EXPRESSIONS

The mode of an expression may be absolute, relocatable in the current program section, relocatable in another program section, or external as defined below.

A term of an expression is absolute, relocatable (current section or other section), or external, correspondingly as its definer (i.e., number, symbol, etc.) is absolute, relocatable (current or other), or external. Numbers, permanent symbols,and generated data are always treated as absolute.

An absolute expression is defined as:

1. Absolute term preceded optionally by a single plus or minus sign or
2. Relocatable expression minus a relocatable term where both items belong to the same program section or
3. Absolute expression followed by an operator followed by an absolute expression.

A relocatable (current or other program section) expression is defined as:

1. Relocatable term, or

2. Relocatable expression followed by an arithmetic operator followed by an absolute expression, or

3. Absolute expression followed by a plus operator followed by a relocatable expression.

An external expression is defined as:

1. External term, or

2. External expression followed by an arithmetic operator followed by an absolute term, or

3. Absolute expression followed by a plus operator followed by an external expression.

In the following examples:

ABS is an absolute symbol
RELC is a relocatable symbol in current program section
RELO is a relocatable symbol in non-current program section
EXT is an external symbol

Examples:

Legal Expressions

```
;EXTERNAL    EXPRESSIONS
    EXT
    EXT -ABS
    ABS +EXT

;RELOCATABLE EXPRESSION

    RELC
    RELC+ABS
    ABS+RELC
    RELC+RELC-RELC
    RELO
    RELO-ABS
    ABS+RELO
    RELO+RELO-RELO
    RELO-RELO+RELO

;ABSOLUTE    EXPRESSION
    RELC-RELC
    RELO-RELO
    ABS+RELC-RELC&ABS
    ABS+RELO-RELO!ABS
```

Illegal Expressions

```
EXT+RELC
EXT+RELO
RELC+RELC
RELO+RELO
ABS-RELC
RELC-RELO
RELO-EXT
RELO!RELO
RELC&RELC
```

CHAPTER 5

ASSEMBLY LOCATION COUNTER

The period (.) is the symbol for the assembly location counter. When used in the operand field of an instruction, it represents the address of the first word of the instruction. When used in the operand field of an assembler directive, it represents the address of the current byte or word. For example:

```
A:  MOV #.,RØ      ;. REFERS TO LOCATION A,
                   ;I.E., THE ADDRESS OF THE
                   ;MOV INSTRUCTION.
```

(# is explained in Section 7.9).

At the beginning of each assembly pass, the Assembler clears the location counter. Normally, consecutive memory locations are assigned to each byte of object data generated. However, the location where the object data is stored may be changed by a direct assignment altering the location counter:

```
.=expression
```

Similar to other symbols, the location counter symbol . has a mode associated with it. However, the mode cannot be external. Neither can one change the existing mode of the location counter by using a defining expression of a different mode.

The mode of the location counter symbol can be changed by the use of the .ASECT or .CSECT directive as explained in Section 8.3.

The expression defining the location counter must not contain forward references or symbols that vary from one pass to another.

Examples:

```
.ASECT
    .=5ØØ                  ;SET LOCATION COUNTER TO ABSOLUTE
                           ;500
```

```
FIRST:    MOV   .+10,COUNT   ;THE LABEL FIRST HAS THE VALUE 500(8)
                             ;.+10 EQUALS 510(8). THE CONTENTS OF
                             ;THE LOCATION 510(8) WILL BE DEPOSITED
                             ;IN LOCATION COUNT.

    .=520                    ;THE ASSEMBLY LOCATION COUNTER NOW
                             ;HAS A VALUE OF ABSOLUTE 520(8).

SECOND:   MOV   .,INDEX      ;THE LABEL SECOND HAS THE VALUE 520(8).
                             ;THE CONTENTS OF LOCATION 520(8), THAT
                             ;IS, THE BINARY CODE FOR THE INSTRUC-
                             ;TION ITSELF, WILL BE DEPOSITED IN
                             ;LOCATION INDEX.

.CSECT
    .=.+20                   ;SET LOCATION COUNTER TO RELOCATABLE
                             ;20 OF THE UNNAMED PROGRAM SECTION.

THIRD:    .WORD 0            ;THE LABEL THIRD HAS THE VALUE OF
                             ;RELOCATABLE 20.
```

Storage area may be reserved by advancing the location counter. For example, if the current value of the location counter is 1000, the direct assignment statement

```
.=.+100
```

will reserve 100_8 bytes of storage space in the program. The next instruction will be stored at 1100.

CHAPTER 6

RELOCATION AND LINKING

The output of the relocatable assembler is an object module which must be processed by the Link-11 Linker before loading and execution. (See PDP-11 Link-11 Linker, Programmer's Manual, DEC-11-ZLDA-D for details.) The Linker essentially fixes (i.e., makes absolute) the values of external or relocatable symbols and creates another module (load module) which contains the binary data to be loaded and executed.

To enable the Linker to fix the value of an expression, the Assembler issues certain directives to the Linker together with the required parameters. In the case of relocatable expressions the Linker adds the base of the associated relocatable section (the location in memory of relocatable 0) to the value of the relocatable expression provided by the Assembler. In the case of an external expression the value of the external term in the expression is determined by the Linker (since the external symbol must be defined in one of the other object modules which are being linked together) and adds it to the value of the external expression provided by the assembler.

All instructions that are to be modified as described above will be marked by an apostrophe in the assembly listing. Thus the binary text output will look as follows for the given examples:

```
005065'  CLR  EXTERNAL(5)
000000                          ;VALUE OF EXTERNAL SYMBOL
                                ;ASSEMBLED ZERO; WILL BE
                                ;MODIFIED BY THE LINKER.

005065'  CLR  EXTERNAL+6(5)     ;THE ABSOLUTE PORTION OF THE
000006                          ;EXPRESSION (000006) IS ADDED
                                ;BY THE LINKER TO THE VALUE
                                ;OF THE EXTERNAL SYMBOL

005065'  CLR RELOCATABLE(5)     ;ASSUMING WE ARE IN THE ABSOLUTE
000040                          ;SECTION AND THE VALUE OF RELOCAT-
                                ;ABLE IS RELOCATABLE 40.
```

CHAPTER 7

ADDRESSING

The program counter (register 7 of the eight general registers) always contains the address of the next word to be fetched; i.e., the address of the next instruction to be executed, or the second or third word of the current instruction.

In order to understand how the address modes operate and how they assemble, the action of the program counter must be understood. The key rule is:

> Whenever the processor implicitly uses the program counter to fetch a word from memory, the program counter is automatically incremented by two after the fetch.

That is, when an instruction is fetched, the PC is incremented by two, so that it is pointing to the next word in memory; and, if an instruction uses indexing (Sections 7.7, 7.8 and 7.10) the processor uses the program counter to fetch the base from memory. Hence, using the rule above, the PC increments by two, and now points to the next word.

The following conventions are used in this section.

a. Let E be any expression as defined in Chapter 4.

b. Let R be a register expression. This is any expression containing a term preceded by a % character or a symbol previously equated to such a term.

   Examples:

```
R0 = %0          ;GENERAL REGISTER 0
R1 = R0+1        ;GENERAL REGISTER 1
R2 = 1+%1        ;GENERAL REGISTER 2
```

c. Let ER be a register expression or an expression in the range 0 to 7 inclusive.

d. Let A be a general address specification which produces a 6-bit mode address field as described in Section I, Chapter 2 of the PDP-11 Handbook 1971.

The addressing specifications, A, may now be explained in terms of E, R, and ER as defined above. Each will be illustrated with the single operand instruction CLR or double operand instruction MOV.

7.1 REGISTER MODE

The register contains the operand.

```
Format:     R
Example:    R0=%0           ;DEFINE R0 AS REGISTER 0
            CLR R0          ;CLEAR REGISTER 0
```

7.2 DEFERRED REGISTER MODE

The register contains the address of the operand.

```
Format:     @R or (ER)
Example:    CLR @R1         ;CLEAR THE WORD AT THE
              or            ;ADDRESS CONTAINED IN
            CLR (1)         ;REGISTER 1
```

7.3 AUTOINCREMENT MODE

The contents of the register are incremented immediately after being used as the address of the operand. (See note below.)

```
Format:     (ER)+
Examples:   CLR  (R0)+      ;CLEAR WORDS AT ADDRESSES CON-
            CLR  (R0+3)+    ;TAINED IN REGISTERS 0, 3 AND 2
            CLR  (2)+       ;AND INCREMENT REGISTER CONTENTS
                            ;BY TWO.
```

NOTE

Both JMP and JSR instructions using non-deferred autoincrement mode, autoincrement the register before its use.

In double operand instructions of the addressing form %R,(R)+ or %R,-(R) where the source and destination registers are the same, the source operand is evaluated as the autoincremented or autodecremented value; but the destination register, at the time it is used, still contains the originally intended effective address. For example, if Register 0 contains 100, the following occurs:

```
MOV R0,(0)+     ;The quantity 102 is moved
                ;to location 100
MOV R0,-(0)     ;The quantity 76 is moved
                ;to location 76.
```

The use of these forms should be avoided, as they are not guaranteed to remain in future PDP-11's.

7.4 DEFERRED AUTOINCREMENT MODE

The register contains the pointer to the address of the operand. The contents of the register are incremented after being used.

```
Format:     @(ER)+
Example:    CLR @(3)+       ;CONTENTS OF REGISTER 3 POINT TO
                            ;ADDRESS OF WORD TO BE CLEARED
                            ;BEFORE BEING INCREMENTED BY TWO.
```

7.5 AUTODECREMENT MODE

The contents of the register are decremented before being used as the address of the operand (see note at top of page).

```
Format:     -(ER)
Examples:   CLR -(R0)       ;DECREMENT CONTENTS OF REGISTERS
            CLR -(R0+3)     ;0, 3, AND 2 BY TWO BEFORE USING
            CLR -(2)        ;AS ADDRESSES OF WORDS TO BE
                            ;CLEARED.
```

7.6 DEFERRED AUTODECREMENT MODE

The contents of the register are decremented before being used as the pointer to the address of the operand.

```
Format:     @-(ER)
Example:    CLR @-(2)       ;DECREMENT CONTENTS OF REG. 2 BY
                            ;TWO BEFORE USING AS POINTER TO
                            ;ADDRESS OF WORD TO BE CLEARED.
```

7.7 INDEX MODE

Format: E(ER)

The value o an expression E is stored as the second or third word of the instruction. The effective address is calculated as the value of E plus the contents of register ER. The value E is called the base.

Examples:

```
CLR X+2(R1)      ;EFFECTIVE ADDRESS IS X+2 PLUS
                 ;THE CONTENTS OF REGISTER 1.

CLR  -2(3)       ;EFFECTIVE ADDRESS IS -2 PLUS
                 ;THE CONTENTS OF REGISTER 3.
```

7.8 DEFERRED INDEX MODE

An expression plus the contents of a register gives the pointer to the address of the operand.

Format: @E(ER)

```
Example: CLR @14(4)      ;IF REGISTER 4 HOLDS 100, AND
                         ;LOCATION 114 HOLDS 2000, LOC.
                         ;2000 IS CLEARED.
```

7.9 IMMEDIATE MODE AND DEFERRED IMMEDIATE (ABSOLUTE) MODE

The immediate mode allows the operand itself to be stored as the second or third word of the instruction. It is assembled as an autoincrement of register 7, the PC.

Format: #E

Examples:

```
MOV #100, R0     ;MOVE AN OCTAL 100 TO REGISTER 0
MOV  #X, R0      ;MOVE THE VALUE OF SYMBOL X TO
                 ;REGISTER 0.
```

The operation of this mode is explained as follows:

The statement MOV #100,R3 assembles as two words. These are:

```
0 1 2 7 0 3
0 0 0 1 0 0
```

Just before this instruction is fetched and executed, the PC points to the first word of the instruction. The processor fetches the first word and increments the PC by two. The source operand mode is 27 (autoincrement the PC). Thus the PC is used as a pointer to fetch the operand (the second word of the instruction) before being incremented by two, to point to the next instruction.

If the #E is preceded by @, E specifies an absolute address.

7.10 RELATIVE AND DEFERRED RELATIVE MODES

Relative mode is the normal mode for memory references.

```
Format:   E
Examples:
          CLR  100        ;CLEAR LOCATION 100
          MOV  X,Y        ;MOVE CONTENTS OF LOCATION X TO
                          ;LOCATION Y.
```

This mode is assembled as Index mode, using 7, the PC, as the register. The base of the address calculation, which is stored in the second or third word of the instruction, is not the address of the operand. Rather, it is the number which, when added to the PC, becomes the address of the operand. Thus, the base is X-PC. The operation is explained as follows:

If the statement MOV 100,R3 is assembled at absolute location 20 then the assembled code is:

```
Location 20:          0 1 6 7 0 3
Location 22           0 0 0 0 5 4
```

The processor fetches the MOV instruction and adds two to the PC so that it points to location 22. The source operand mode is 67; that is, indexed by the PC. To pick up the base, the processor fetches the word pointed to by the PC and adds two to the PC. The PC now points to location 24. To calculate the address of the source operand, the base is added to the designated register. That is, BASE+PC=54+24=100, the operand address.

Since the Assembler considers "." as the address of the first word of the instruction, an equivalent statement would be

```
MOV 100-.-4(PC),R3
```

This mode is called relative because the operand address is calculated relative to the current PC. The base is the distance (in bytes) between the operand and the current PC. If the operator and its operand are moved in memory so that the distance between the operator and data remains constant, the instruction will operate correctly.

If E is preceded by @, the expression's value is the pointer to the address of the operand.

7.11 TABLE OF MODE FORMS AND CODES

Each instruction takes at least one word. Operands of the first six forms listed below, do not increase the length of an instruction. Each operand in one of the other modes, however, increases the instruction length by one word.

Form	Mode	Meaning
R	0n	Register
@R or (ER)	1n	Register deferred
(ER)+	2n	Autoincrement
@(ER)+	3n	Autoincrement deferred
-(ER)	4n	Autodecrement
@-(ER)	5n	Autodecrement deferred

None of the above forms increases the instruction length.

E (ER)	6n	Index
@E(ER)	7n	Index deferred
#E	27	Immediate
@#E	37	Absolute memory reference
E	67	Relative
@E	77	Relative deferred reference

Any of the above forms adds a word to the instruction length.

NOTES

1. An alternate form for @R is (ER). However, the form @(ER) is equivalent to @0(ER).

2. The form @#E differs from the form E in that the second or third word of the instruction contains the absolute address of the operand rather than the relative distance between the operand and the PC. Thus, the instruction CLR @#100 will clear location 100 even if the instruction is moved from the point at which it was assembled.

7.12 INSTRUCTION FORMS

The instruction mnemonics are given in Appendix B. This section defines the number and nature of the operand fields for these instructions.

In the table that follows, let R, E, and ER represent expressions as defined on page 7-1, and let A be a 6-bit address specification of any one of the forms:

E	@E
R	@R or (R)
(ER)+	@(ER)+
-(ER)	@-(ER)
E(ER)	@E(ER)
#E	@#E

Table 2.1 Instruction Operand Fields

Instruction	Form	Example
Double Operand	Op A,A	MOV (R6)+,@Y
Single Operand	Op A	CLR -(R2)
Operate	Op	HALT
Branch	Op E	BR X+2
		BLO .-4
	where $-128 \leq (E-.-2)/2 \leq 127_{10}$	
Subroutine Call	JSR ER,A	JSR PC,SUBR
Subroutine Return	RTS ER	RTS PC
EMT/TRAP	Op or OP E	EMT
	where $0 \leq E \leq 377_8$	EMT 31

The branch instructions are one word instructions. The high byte contains the op code and the low byte contains an 8-bit signed offset (7 bits plus sign) which specifies the branch address relative to the PC. The hardware calculates the branch address as follows:

a) Extend the sign of the offset through bits 8-15.

b) Multiply the result by 2. This creates a word offset rather than a byte offset.

c) Add the result to the PC to form the final branch address.

The Assembler performs the reverse operation to form the byte offset from the specified address. Remember that when the offset is added to the PC, the PC is pointing to the word following the branch instruction; hence the factor -2 in the calculation.

Byte offset = (E-PC)/2 truncated to eight bits.

Since PC = .+2, we have

Byte offset = (E-.-2)/2 truncated to eight bits.

NOTE

It is illegal to branch to a location specified as an external symbol, or to a relocatable symbol when within an absolute section, or to an absolute symbol or a relocatable symbol of another program section when within a relocatable section.

The EMT and TRAP instructions do not use the low-order byte of the word. This allows information to be transferred to the trap handlers in the low-order byte. If EMT or TRAP is followed by an expression, the value is put into the low-order byte of the word. However, if the expression is too big ($>377_8$) it is truncated to eight bits and a Truncation (T) error occurs.

CHAPTER 8

ASSEMBLER DIRECTIVES

Assembler directives (sometimes called pseudo-ops) direct the assembly process and may generate data.

Assembler directives may be preceded by a label and followed by a comment. The assembler directive occupies the operator field. Only one directive may be placed in any one statement. One or more operands may occupy the operand field or it may be void -- allowable operands vary from directive to directive.

8.1 .TITLE

The .TITLE directive is used to name the object module. The name assigned is the first symbol following the directive. If there is no .TITLE statement the default name assigned is ".MAIN.". If there is more than one .TITLE directive, only the last one encountered is operative.

8.2 .GLOBL

The .GLOBL directive is used to declare a symbol as being global. This is the mechanism by which separately assembled object modules communicate with one another. It may be an entry symbol in which case it is defined in the program or it may be an external symbol in which case it should be defined in another program which will be linked with this program by Link-11.

The form of the .GLOBL directive is

```
.GLOBL  NAMA,NAMB,...,NAMN
```

NOTE

A symbol cannot be declared global by assigning it to a global expression in a direct assignment statement.

8.3 PROGRAM SECTION DIRECTIVES

```
.ASECT
.CSECT
.CSECT  symbol
```

The Assembler provides for 255_{10} program sections; an absolute section declared by .ASECT, an unnamed relocatable program section declared by .CSECT, and 253_{10} named relocatable program sections declared by .CSECT symbol, where symbol is any legal symbolic name. These directives allow the user to:

1. Create his program (object module) in sections:

   The Assembler maintains separate location counters for each section. This allows the user to write statements which are not physically contiguous but will be loaded contiguously. The following examples will clarify this:

```
    .CSECT              ;START THE UNNAMED RELOCATABLE SECTION
A:  0                   ;ASSEMBLED AT RELOCATABLE 0,
B:  0                   ;     RELOCATABLE 2 AND
C:  0                   ;     RELOCATABLE 4,
ST: CLR A               ;ASSEMBLE CODE AT
    CLR B               ;     RELOCATABLE ADDRESS
    CLR C               ;     6 THROUGH 21
    .ASECT              ;START THE ABSOLUTE SECTION
    .=4                 ;ASSEMBLE CODE AT
    .WORD .+2,HALT      ;ABSOLUTE 4 THROUGH 7,
    .CSECT              ;RESUME THE UNNAMED RELOCATABLE
                        ;   SECTION
    INC A               ;ASSEMBLE CODE AT
    BR ST               ;   RELOCATABLE 22 THROUGH 27,
    .END
```

   The first appearance of .CSECT or .ASECT assumes the location counter is at relocatable or absolute zero, respectively. The scope of each directive extends until a directive to the contrary is given. Further occurrences of the same .CSECT or .ASECT resume assemblying where the section was left off.

```
        .CSECT  COM1        ;DECLARE SECTION COM1
A:      0                   ;ASSEMBLED AT RELOCATABLE 0,
B:      0                   ;ASSEMBLED AT RELOCATABLE 2,
C:      0                   ;ASSEMBLED AT RELOCATABLE 4,
        .CSECT  COM2        ;DECLARE SECTION COM2
X:      0                   ;ASSEMBLED AT RELOCATABLE 0,
Y:      0                   ;ASSEMBLED AT RELOCATABLE 2,
        .CSECT  COM1        ;RETURN TO COM1
D:      0                   ;ASSEMBLED AT RELOCATABLE 6,
        .END
```

The Assembler automatically begins assemblying at relocatable zero of the unnamed .CSECT if not instructed otherwise; that is, the first statement of an assembly is an implied .CSECT.

All labels in an absolute section are absolute; all labels in a relocatable section are relocatable. The location counter symbol, ".", is relocatable or absolute when referenced in a relocatable or absolute section, respectively. Undefined internal symbols are assigned the value of relocatable or absolute zero in a relocatable or absolute section, respectively. Any labels appearing on a .ASECT or .CSECT statement are assigned the value of the location counter before the .ASECT or .CSECT takes effect. Thus, if the first statement of a program is:

```
A:      .ASECT
```

Then A is assigned to relocatable zero and is associated with the unnamed relocatable section (because the Assembler implicitly begins assembly in the unnamed relocatable section).

Since it is not known at assembly time where the program sections are to be loaded, all references between sections in a single assembly are translated by the Assembler to references relative to the base of that section. The Assembler provides the Linker with the necessary information to resolve the linkage.

Note that this is not necessary when making a reference to an absolute section (the Assembler knows all load addresses of an absolute section).

Examples:

```
        .ASECT
        .=1000
A:      CLR X              ;ASSEMBLED AS CLR BASE OF UNNAMED
                           ;   RELOCATABLE SECTION + 10
        JMP Y              ;ASSEMBLED AS JMP BASE OF UNNAMED
                           ;   RELOCATABLE SECTION + 6
        .CSECT
        MOV R0,R1
        JMP A              ;ASSEMBLED AS JMP 1000
Y:      HALT
X:      0
        .END
```

In the above example the references to X and Y were translated into references relative to the base of the unnamed relocatable section.

2. Share code and/or data between object modules (separate assemblies):

Named relocatable program sections operate as FORTRAN labeled COMMON; that is, sections of the same name from different assemblies are all loaded at the same location by Link-11. The unnamed relocatable section is the exception to this as all unnamed relocatable sections are loaded in unique areas by Link-11.

Note that there is no conflict between internal symbolic names and program section names; that is, it is legal to use the same symbolic name for both purposes. In fact, considering FORTRAN again, this is a necessity to accommodate the FORTRAN statement

```
COMMON /X/A,B,C,X
```

Where the symbol X represents the base of this program section and also the 4th element of this program section.

Also, there is no conflict between program section names and .GLOBL names. In FORTRAN language, COMMON block names and SUBROUTINE names may be the same.

8.4 .EOT

Under the Disk Operating System, the .EOT directive is effectively ignored. The physical End Of Tape itself allows several physically separate tapes to be assembled as one program. The last tape should be terminated by a .END directive; however, in its absence, the TAPES command string switch (Section 9.2) may be used.

8.5 .EVEN

The .EVEN directive ensures that the assembly location counter is even by adding one of it is odd. Any operands following a .EVEN directive will be ignored.

8.6 .END

The .END directive indicates the physical end of the source program. The .END directive may be followed by only one operand, which must be non-global, an expression indicating the program's transfer address.

At load time, the load module will be loaded and program execution will begin at the transfer address indicated by the .END directive. If the address is not specified, the loader will halt after reading in the load module, and an F376 error message will be printed on the teleprinter. The user can return to the DOS Monitor and use the GET and BEGIN commands to execute the user program.

8.7 .WORD

The .WORD assembler directive may have one or more operands, separated by commas. Each operand is stored in a word of the object program. If there is more than one operand, they are stored in successive words. The operands may be any legally formed expression. For example:

```
.=1420
SAL=0
.WORD 177535,.+4,SAL          ;STORED IN WORDS 1420, 1422, AND
                              ;1424 WILL BE 177535, 1426, AND 0.
```

Values exceeding 16 bits will be truncated from the left to word length.

A .WORD directive followed by one or more void operands separated by commas will store zeros for the void operands. For example,

```
.=1430          ;ZERO, FIVE, AND ZERO ARE STORED
.WORD ,5,       ;IN WORDS 1430, 1432, AND 1434.
```

An operator field left blank will be interpreted as the .WORD directive if the operand field contains one or more expressions. The first term of the first expression in the operand field must not be an instruction mnemonic or assembler directive unless preceded by a +, -, or one of the logical operators, ! or &. For example,

```
.=440                         ;THE OP-CODE FOR MOV, WHICH
                              ;    IS 010000,
LABEL:      +MOV,LABEL        ;IS STORED IN LOCATION 440. 440 IS
                              ;    STORED IN LOCATION 442.
```

Note that the default .WORD will occur whenever there is a leading arithmetic or logical operator, or whenever a leading symbol is encountered which is not recognized as an instruction mnemonic or assembler directive. Therefore, <u>if an instruction mnemonic or assembler directive is misspelled, the .WORD directive is assumed and errors will result.</u> Assume that MOV is spelled incorrectly as MOR:

```
MOR  A,B
```

Two error codes can result: Q will occur because an expression operator is missing between MOR and A, and a U will occur if MOR is undefined. Two words will be generated: one for MOR A and one for B.

8.8 .BYTE

The .BYTE assembler directive may have one or more operands separated by commas. Each operand is stored in a byte of the object program. If multiple operands are specified, they are stored in successive bytes. The operands may be any legally formed expression with a result of 8 bits or less. For example:

```
SAM=5                   ;STORED IN LOCATION 410 WILL BE
.=410                   ;060 (THE OCTAL EQUIVALENT OF 48).
.BYTE 48.,SAM           ;IN 411 WILL BE 005.
```

If the expression has a result of more than 8 bits, it will be truncated to its low-order 8 bits and will be flagged as a T error. If an operand after the .BYTE directive is left void, it will be interpreted as zero. For example:

```
.=420           ;ZERO WILL BE STORED IN
.BYTE , ,       ;BYTES 420, 421 and 422.
```

If the expression is relocatable, a warning flag, A, will be given, since at link time it is very likely that the relocation will result in an expression of more than 8 bits.

8.9 .ASCII

The .ASCII directive translates strings of ASCII characters into their 7-bit ASCII codes, with the exception of null, rubout, carriage return, line feed, and form feed. The text to be translated is delimited by a character at the beginning and the end of the text. The delimiting character may be any printing ASCII character except colon and equal sign and those used in the text string. The 7-bit ASCII code generated for each character will be stored in successive bytes of the object program. For example:

```
.=500
.ASCII      /YES/           ;THE ASCII CODE FOR "Y" WILL BE
                            ;STORED IN 500, THE CODE FOR "E"
                            ;IN 501, THE CODE FOR "S" IN 502.

.ASCII      /5+3/2/         ;THE DELIMITING CHARACTER OCCURS
                            ;AMONG THE OPERANDS. THE ASCII
                            ;CODES FOR "5", "+", AND "3" ARE
                            ;STORED IN BYTES 503,504, AND
                            ;505. 2/ IS NOT ASSEMBLED.
```

The .ASCII directive may be terminated by any legal terminator.

8.10 .RAD50

PDP-11 systems programs often handle symbols in a specially coded form called RADIX 50 (this form is sometimes referred to as MOD40). This form allows 3 characters to be packed into 16 bits; therefore, any 6-character symbol can be held in two words. The form of the directive is:

```
.RAD50      /CCC/
```

The single operand is of the form /CCC/ where the slash (the delimiter) can be any printable character except for = and : . The delimiters enclose the characters to be converted which may be A through Z, 0 through 9, dollar ($), dot (.) and space ( ). If there are fewer than 3 characters they are considered to be left justified and trailing spaces are assumed.

Non-printing characters are replaced with a ? character and cause an I error flag to be set. Illegal printing characters set the Q end flag.

Examples:

```
.RAD50      /ABC/           ;PACK ABC INTO ONE WORD
.RAD50      /AB/            ;PACK AB (SPACE) INTO ONE WORD
.RAD50      //              ;PACK 3 SPACES INTO ONE WORD
```

The packing algorithm is as follows:

A. Each character is translated into its RADIX 50 equivalent as indicated in the following table:

Character	RADIX 50 Equivalent (octal)
(space)	0
A-Z	1-32
$	33
.	34
0-9	36-47

Note that another character could be defined for code 35.

B. The RADIX 50 equivalents for characters 1 through 3 (C1,C2,C3) are combined as follows:

```
RESULT=((C1*50)+C2)*50+C3
```

8.11 .LIMIT

A program often wishes to know the boundaries of the load module's relocatable code. The .LIMIT directive generates two words into which the Linker puts the low and high addresses of the relocated code. The low address (inserted into the first word) is the address of the first byte of code. The high address is the address of the first free byte following the relocated code. These addresses will always be even since all relocatable sections are loaded at even addresses and if a relocatable section consists of an odd number of bytes the Linker adds one to the size to make it even.

8.12 CONDITIONAL ASSEMBLY DIRECTIVES

Conditional assembly directives provide the programmer with the capability to conditionally include or not include portions of his source code in the assembly process. In what follows, E denotes an expression and S(I) denotes a symbol. The conditional directives are:

```
.IFZ  or  .IFEQ  E          IF E=0
.IFNZ or  .IFNE  E          IF E≠0
.IFL  or  .IFLT  E          IF E<0
.IFLE            E          IF E≤0
.IFG  or  .IFGT  E          IF E>0
.IFGE            E          IF E≥0
.IFDF            S(1)[!,&]S(2)[!,&]...[!,&]S(N)     (!=OR, &=AND)
.IFNDF           S(1)[!,&]S(2)[!,&]...[!,&]S(N)
```

If the condition is met, all statements up to the matching .ENDC are assembled. Otherwise, the statements are ignored until the matching .ENDC is detected.

In the above, .IFDF and .IFNDF mean "if defined" and "if undefined" respectively. The scan is left to right, no parentheses permitted.

Example:

.IFDF S!T&U	Means assemble if either S or T is defined and U is defined
.IFNDF T&U!S	Means assemble if both T and U are undefined or if S is undefined

General Remarks:

An errored or null expression takes the default value 0 for purposes of the conditional test. An error in syntax, e.g., a terminator other than ;, !, &, or CR results in the undefined situation for .IFDF and .IFNDF, as does an errored or null symbol.

All conditionals must end with the .ENDC directive. Anything in the operand field of .ENDC is ignored. Nesting is permitted up to a depth of 127_{10}. Labels are permitted on conditional directives, but the scan is purely left to right. For example:

```
        .IFZ 1
A:      .ENDC
```

A is ignored.

```
A:      .IFZ 1
        .ENDC
```

A is entered in the symbol table.

If an .END is encountered while inside a satisfied conditional, a Q flag will appear, but the .END directive will still be processed normally. If more .ENDC's appear than are required, Q flags appear on the extras.

CHAPTER 9

OPERATING PROCEDURES

The Assembler enables you to assemble into a relocatable binary file (object module) an ASCII source containing PAL-11R statements. To do this, two or three passes are necessary. On the first pass, the Assembler creates a table of user-defined symbols and their associated values, and a list of undefined symbols is printed on the command output device. On subsequent passes the Assembler assembles the program and produces the outputs specified during the initial dialogue (Section 9.2). The Assembler initiates the dialogue immediately after being loaded and after the last pass of an assembly.

9.1 LOADING PAL-11R

PAL-11R is loaded via the Disk Monitor command RUN as follows:

```
RUN PAL11R
```

9.2 INITIAL DIALOGUE

When the Assembler is ready to accept the user's command string, it outputs the following lines to the command device:

```
PAL11R  VØØ5A
#
```

The user must now type his command string on the same line as the #.

If an error is made in typing at any time, typing the RUBOUT key will erase the immediately preceding character if it is on the current line. Typing CTRL/U will erase the entire line.

The format of the command string adheres to the requirements of the Disk Operating System's (DOS) Command String Interpreter (CSI). The Assembler's file specifications must appear in the following order:

```
# Binary, Listing, Symbol Table < Source 1, Source 2
```

A null specification field signifies that the associated input or output type is not desired. Each specification contains the following information:

```
dev:filnam.ext[uic]/PASS:value/TAPES:value
```

If a syntactical error is detected in the command string, the Assembler will output the command string up to and including the point where the error was detected followed by the character "?". The Assembler will then output # again, and the user must retype his entire command.

The following command string semantic errors will be detected:

		Error Code
1.	Illegal switch Too many switches Illegal switch value Too many switch values	S203
2.	Too many output file specifications	S204
3.	Too many input file specifications	S205
4.	Input file missing	S206

After outputting the appropriate error code, the Assembler will output # again and the user must retype his entire command.

The default value for each specification is noted below:

	DEV	FILNAM	EXT	UIC	PASS	TAPES
Source 1	system device	None	PAL	This User	All Passes***	1
Source 2	system device*	None	PAL	This User	All Passes	1
Binary	system device	Same as Source** FILNAM	OBJ	This User	2	
Listing	system device*	Same as Source** FILNAM	LST	This User	2	
Symbols	system device*	Same as Source** FILNAM	SYM	This User	End of Pass 1	

* Or last device specified on the input side of the <.

** The default FILNAM is Source 2's if this field is present; otherwise, Source 1's is used.

*** If /PASS:n (where 0 <n <4) is in the Source 1 field, this file will only be accessed during pass 1. This is useful for parameter assignments associated with conditional assemblies.

Source 1 and Source 2 cannot both be associated with the same unit file device such as the paper tape reader. This will result in a Monitor error.

The TAPES switch is recognized in the two input fields of PAL-11R's command string; it is ignored in the output fields. The value associated with the switch specifies the maximum decimal number of tapes* that will be processed from the designated device. The .END assembly directive always terminates an assembly pass; however, if it is missing, the maximum tape count will effect termination. In the latter case, the error line count output by the Assembler is incremented by one to warn the user of the omission. The default number of tapes is 1. The following examples should clarify the usages of the TAPES switch.

1. #PP:,LP:,LP:<PR:/TAPES:5

   All passes of the assembly will terminate either when a) the .END directive is encountered or b) five tapes have been input from the reader.

2. #PP:,LP:<PR:

   All passes of the assembly will terminate when one tape has been input from the reader.

3. #PP:<PR:/PASS:1/TAPES:2,DT1:FILE

   Two parameter tapes will be input from the reader during pass 1 only. All passes of the assembly will terminate either when a) the .END directive is encountered or b) the source file FILE.PAL has been input from DT1.

4. #PP:<DT1:PARAM/PASS:1,PR:/TA:4

   One parameter file, PARAM.PAL, will be input from DT1 during pass 1 only. All passes of the assembly will terminate either when a) the .END directive is encountered or b) four tapes have been input from the reader.

---

* This switch is most applicable to non-file oriented devices such as the paper tape reader. However, it would cause the same file on DECtape to be processed the specified number of times during each pass.

The 37 switch is recognized in all the fields of PAL-11R's command string. Its presence directs the Assembler to assemble all relative addresses (address mode 67) as absolute addresses (address mode 37).

e.g., CLR ADDR normally assembles as

```
ØØ5Ø67
  N
```

where N is the offset between ADDR and the current PC.

When the 37 switch has been encountered in the command string

CLR ADDR assembles as

```
ØØ5Ø37
 ADDR
```

just as if the source code read

```
CLR  @#ADDR
```

This switch is useful during the debugging phase of program development as it facilitates tracing through assembly listings. Note that when the 37 switch is used, position independent coding is not possible.

The following example command string shows the 37 switch specification

```
#PP:,LP:/37<DT1:FILE
```

Note that DOS only allows one output file to be open on a DECtape at a time. The Assembler loosens this restriction by:

1. Providing the PASS switch, for example:

```
#DTA1:,DTA1:/PASS:3<TEST
```

This command causes the input file TEST.PAL on the system device to be assembled with the binary output, TEST.OBJ, going to DTA1 during pass 2 and the listing output, TEST.LST, going to DTA1 during pass 3.

2. Closing files and releasing datasets[1] as soon as possible, for example:

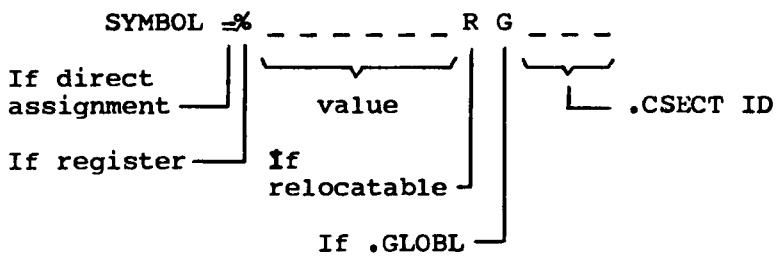
```
#,DTA1:,DTA1:/PASS:2<TEST
```

---

1 The releasing of datasets has the additional advantage that all drivers and buffers do not have to be core resident during the entire assembly.

This command causes the input file TEST.PAL on the system device to be assembled with no binary output; the listing output (TEST.LST) goes to DTA1 during pass 2 and the symbol table output (TEST.SYM) goes to DTA1 at the end of pass 2.

The symbol table is output alphabetically, three symbols per line. Each symbol is output in the following format:



If the symbol is undefined, six asterisks replace its value. The identifying characters indicate the class of the symbol; that is, whether it is a label, direct assignment, register symbol, relocatable, absolute, or global.

The .CSECT ID field is left blank for .ASECT symbols and unnamed .CSECT symbols as they can be identified by the absence or presence of the R , respectively.

Symbols belonging to named .CSECT's have the ID's 002 through 376_8 associated with them, where the nth named .CSECT encountered will have the ID n+1.

Immediately following the symbol table listing will be a listing of the .CSECT names, their sizes in bytes, and their corresponding ID's. There will be one entry per line in the following format:

```
FIRST       ØØ4Ø2Ø     ØØ2
SECOND      ØØØ376     ØØ3
```

.
.
.

When no outputs are omitted and the default passes are in effect, the Assembler performs as follows:

PASS 1: Assembler creates a table of user-defined symbols and their associated values to be used in assembling the source to object program. Undefined symbols (not including external .GLOBL's) are listed on the command output device at the end of the pass. The symbol table is also output at this time.

PASS 2: Assembler outputs the object module, and the listing file which includes the pass error line count (in octal), and prints the pass error line count on the command output device.

9.3 ASSEMBLY DIALOGUE

During assembly, the Assembler may output various messages to the command device. The message may be:

1. informative,
2. request for operator action, or
3. terminal.

The operator may also initiate a message. For example, CTRL/C and RESTART at any time, to stop the assembly process and restart the initial dialogue as mentioned in the previous section.

At the end of each pass (except the last) when the .END assembly directive is encountered, END is output to the command device. For file structured input devices, the next pass is automatically begun. For non-file structured input devices, e.g., paper tape reader, the Disk Operating System will announce that the reader must be reloaded prior to continuing:

```
A002    063320
$
```

A002 denotes device not ready, and
063320 is the radix 50 notation for PR.

After the user has reloaded the reader, he should type the Monitor command CONTINUE on the command input device.

This continuing from device-not-ready is also applicable when .EOT is encountered during multiple tape assemblies and when any I/O device requires servicing for such purposes as:

a. line printer paper
b. write enable DECtape
c. tape for punch.

The following terminal errors are printed on the command device. An attempt is made to close all output files and delete the binary file, if open. The Assembler then outputs # and waits for further commands:

S200	more than 376_8 .CSECT's
S201	more than 177_8 nested conditionals
S202	binary or listing file structured device full or input command string too long
S207	input .TRAN error on Assembler's overlay file

RESTART capability:

The assembly may be aborted at any time by typing CTRL/C followed by either the Monitor command RESTART or the Monitor command BEGIN on the command device. All open output files will be closed and released. If the binary file was open, it is deleted. The Assembler will output # and wait for the next command string. It does this automatically at the normal completion of an assembly.

9.4 ASSEMBLY LISTING

The PAL-11R Assembler produces a side-by-side assembly listing of symbolic source statements, their octal equivalents, assigned addresses, and error codes, as follows:

```
EELLLLLL OOOOOOASSS.......S
         OOOOOO
         OOOOOO
```

The E's represent the error field. The L's represent the address. The O's represent the object data in octal. The S's represent the source statement. The A represents an apostrophe which indicates that either the second, third, or both words of the instruction are to be modified by the Linker.

The above represent a three-word statement. The second and third words of the statement are listed under the command word. No addresses precede the second and third words since the address order is sequential.

The third line is omitted for a two-word statement; both second and third lines are omitted for a one-word statement.

For a.BYTE directive, the object data field is three octal digits.

For a direct assignment statement, the value of the defining expression is given in the object code field although it is not actually part of the code of the object program.

The .ASECT and .CSECT directives cause the current value of the appropriate location counter (absolute or relocatable) to be printed.

Each page of the listing is headed by a page number (octal).

9.5 OBJECT MODULE OUTPUT

The output of the Assembler during the binary object pass is an object module which is meaningful only to the Linker. What follows gives an overview of what the object module contains and at what stage each part of it is produced.

The binary object module consists of three main types of data block:

a) Global symbol directory (GSD)
b) Text blocks (TXT)
c) Relocation Directory (RLD)

9.5.1 Global Symbol Directory

As the name suggests, the Global Symbol Directory (GSD) contains a list of all global symbols (entry points and externals) with all entry points associated with the program section symbol in which they reside. Each symbol is in RADIX 50 form and contains

mode information and relative addresses for entry points, and section sizes for program section names. The name of the object module is also in the GSD.

The GSD is created at the start of the binary object pass.

9.5.2. Text Block

The text blocks consist entirely of the binary object data as shown in the listing. The operands are in the unmodified form.

9.5.3 Relocation Directory

The Relocation Directory (RLD) blocks consist of directives to the Linker which may reference the text block preceding it. These directives control the relocation and linking process.

Text and RLD blocks are constructed during the binary object pass. Outputting of each block is done whenever either the Text or RLD buffer is full and whenever the location counter needs to be modified.

CHAPTER 10

ERROR CODES

The error codes printed beside the octal and symbolic code in the assembly listing have the following meanings:

Error Code	Meaning
A	Addressing error. An address within the instruction is incorrect. Also may indicate a relocation error.
B	Bounding error. Instructions or word data are being assembled at an odd address in memory. The location counter is updated by +1.
D	Doubly-defined symbol referenced. Reference was made to a symbol which is defined more than once.
I	Illegal character detected. Illegal characters which are also non-printing are replaced by a ? on the listing.
L	Line buffer overflow, i.e., input line greater than 83 characters. Extra characters on a line, (more than 72_{10}) are ignored.
M	Multiple definition of a label. A label was encountered which was equivalent (in the first six characters) to a previously encountered label.
N	Number containing 8 or 9 has decimal point missing.
P	Phase error. A label's definition or value varies from one pass to another.
Q	Questionable syntax. There are missing arguments or the instruction scan was not completed or a carriage return was not immediately followed by a line feed or form feed.
R	Register-type error. An invalid use of or reference to a register has been made.
T	Truncation error. A number generated more than 16 bits of significance or an expression generated more than 8 bits of significance during the use of the .BYTE directive.
U	Undefined symbol. An undefined symbol was encountered during the evaluation of an expression. Relative to the expression, the undefined symbol is assigned a value of zero.

APPENDIX A

ASCII CHARACTER SET

EVEN PARITY BIT	7-BIT OCTAL CODE	CHARACTER	REMARKS
Ø	ØØØ	NUL	NULL, TAPE FEED, CONTROL/SHIFT/P.
1	ØØ1	SOH	START OF HEADING: ALSO SOM, START OF MESSAGE, CONTROL/A.
1	ØØ2	STX	START OF TEXT; ALSO EOA, END OF ADDRESS, CONTROL/B.
Ø	ØØ3	ETX	END OF TEXT; ALSO EOM, END OF MESSAGE, CONTROL/C.
1	ØØ4	EOT	END OF TRANSMISSION (END); SHUTS OFF TWX MACHINES, CONTROL/D.
Ø	ØØ5	ENQ	ENQUIRY (ENQRY); ALSO WRU, CONTROL/E.
Ø	ØØ6	ACK	ACKNOWLEDGE; ALSO RU, CONTROL/F.
1	ØØ7	BEL	RINGS THE BELL. CONTROL/G.
1	Ø1Ø	BS	BACKSPACE; ALSO FEO, FORMAT EFFECTOR. BACKSPACES SOME MACHINES, CONTROL/H.
Ø	Ø11	HT	HORIZONTAL TAB. CONTROL/I.
Ø	Ø12	LF	LINE FEED OR LINE SPACE (NEW LINE); ADVANCES PAPER TO NEXT LINE, DUPLICATED BY CONTROL/J.
1	Ø13	VT	VERTICAL TAB (VTAB). CONTROL/K.
Ø	Ø14	FF	FORM FEED TO TOP OF NEXT PAGE (PAGE). CONTROL/L.
1	Ø15	CR	CARRIAGE RETURN TO BEGINNING OF LINE. DUPLICATED BY CONTROL/M.
1	Ø16	SO	SHIFT OUT; CHANGES RIBBON COLOR TO RED. CONTROL/N.
Ø	Ø17	SI	SHIFT IN; CHANGES RIBBON COLOR TO BLACK. CONTROL/O.
1	Ø2Ø	DLE	DATA LINK ESCAPE. CONTROL/P (DCØ).
Ø	Ø21	DC1	DEVICE CONTROL 1, TURNS TRANSMITTER (READER) ON, CONTROL/Q (X ON).
Ø	Ø22	DC2	DEVICE CONTROL 2, TURNS PUNCH OR AUXILIARY ON. CONTROL/R (TAPE, AUX ON).
1	Ø23	DC3	DEVICE CONTROL 3, TURNS TRANSMITTER (READER) OFF, CONTROL/S (X OFF).
Ø	Ø24	DC4	DEVICE CONTROL 4, TURNS PUNCH OR AUXILIARY OFF. CONTROL/T (AUX OFF).
1	Ø25	NAK	NEGATIVE ACKNOWLEDGE; ALSO ERR, ERROR. CONTROL/U.
1	Ø26	SYN	SYNCHRONOUS FILE (SYNC). CONTROL/V.
Ø	Ø27	ETB	END OF TRANSMISSION BLOCK; ALSO LEM, LOGICAL END OF MEDIUM. CONTROL/W.
Ø	Ø3Ø	CAN	CANCEL (CANCL). CONTROL/X.
1	Ø31	EM	END OF MEDIUM. CONTROL/Y.
1	Ø32	SUB	SUBSTITUTE. CONTROL/Z.
Ø	Ø33	ESC	ESCAPE. PREFIX. CONTROL/SHIFT/K.
1	Ø34	FS	FILE SEPARATOR. CONTROL/SHIFT/L.

EVEN PARITY BIT	7-BIT OCTAL CODE	CHARACTER	REMARKS
Ø	Ø35	GS	GROUP SEPARATOR. CONTROL/SHIFT/M.
Ø	Ø36	RS	RECORD SEPARATOR. CONTROL/SHIFT/N.
1	Ø37	US	UNIT SEPARATOR. CONTROL/SHIFT/O.
1	Ø4Ø	SP	SPACE.
Ø	Ø41	!	
Ø	Ø42	"	
1	Ø43	#	
Ø	Ø44	$	
1	Ø45	%	
1	Ø46	&	
Ø	Ø47	'	ACCENT ACUTE OR APOSTROPHE.
Ø	Ø5Ø	(	
1	Ø51	)	
1	Ø52	*	
Ø	Ø53	+	
1	Ø54	,	
Ø	Ø55	-	
Ø	Ø56	.	
1	Ø57	/	
Ø	Ø6Ø	Ø	
1	Ø61	1	
1	Ø62	2	
Ø	Ø63	3	
1	Ø64	4	
Ø	Ø65	5	
Ø	Ø66	6	
1	Ø67	7	
1	Ø7Ø	8	
Ø	Ø71	9	
Ø	Ø72	:	
1	Ø73	;	
Ø	Ø74	<	
1	Ø75	=	
1	Ø76	>	
Ø	Ø77	?	
1	1ØØ	@	
Ø	1Ø1	A	
Ø	1Ø2	B	
1	1Ø3	C	
Ø	1Ø4	D	
1	1Ø5	E	
1	1Ø6	F	
Ø	1Ø7	G	
Ø	11Ø	H	
1	111	I	
1	112	J	
Ø	113	K	
1	114	L	
Ø	115	M	
Ø	116	N	
1	117	O	
Ø	12Ø	P	
1	121	Q	
1	122	R	
Ø	123	S	
1	124	T	

EVEN PARITY BIT	7-BIT OCTAL CODE	CHARACTER	REMARKS
Ø	125	U	
Ø	126	V	
1	127	W	
1	13Ø	X	
Ø	131	Y	
Ø	132	Z	
1	133	[	SHIFT/K.
Ø	134	\	SHIFT/L.
1	135	]	SHIFT/M.
1	136	↑	*
Ø	137	←	**
Ø	14Ø	`	ACCENT GRAVE
1	141	a	
1	142	b	
Ø	143	c	
1	144	d	
Ø	145	e	
Ø	146	f	
1	147	g	
1	15Ø	h	
Ø	151	i	
Ø	152	j	
1	153	k	
Ø	154	l	
1	155	m	
1	156	n	
Ø	157	o	
1	16Ø	p	
Ø	161	q	
Ø	162	r	
1	163	s	
Ø	164	t	
1	165	u	
1	166	v	
Ø	167	w	
Ø	17Ø	x	
1	171	y	
1	172	z	
Ø	173	{	
1	174	|	
Ø	175	}	THIS CODE GENERATED BY ALT MODE
Ø	176	~	THIS CODE GENERATED BY ESC KEY (IF PRESENT)
1	177	DEL	DELETE, RUB OUT

* ↑ appears as ^ on some machines.

** ← appears as _ (underscore) on some machines.

APPENDIX B

PAL-11R ASSEMBLY LANGUAGE AND ASSEMBLER

B.1 SPECIAL CHARACTERS

Character	Function
form feed	Source line terminator
line feed	Source line terminator
carriage return	Source statement terminator
:	Label terminator
=	Direct assignment indicator
%	Register term indicator
tab	Item terminator Field terminator
space	Item terminator Field terminator
#	Immediate expression indicator
@	Deferred addressing indicator
(	Initial register indicator
)	Terminal register indicator
,	Operand field separator
;	Comment field indicator
+	Arithmetic addition operator
-	Arithmetic subtraction operator
&	Logical AND operator
!	Logical OR operator
"	Double ASCII character indicator
'	Single ASCII character indicator
.	Assembly location counter

B.2 ADDRESS MODE SYNTAX

n is an integer between 0 and 7 representing a register. R is a register expression, E is an expression, ER is either a register expression or an expression in the range 0 to 7.

Format	Address Mode Name	Address Mode Number	Meaning
R	Register	0n	Register R contains the operand. R is a register expression.
@R or (ER)	Deferred Register	1n	Register R contains the operand address.
(ER)+	Autoincrement	2n	The contents of the register specified by ER are incremented after being used as the address of the operand.
@(ER)+	Deferred Autoincrement	3n	ER contains the pointer to the address of the operand. ER is incremented after use.
-(ER)	Autodecrement	4n	The contents of register ER are decremented before being used as the address of the operand.
@-(ER)	Deferred Autodecrement	5n	The contents of register ER are decremented before being used as the pointer to the address of the operand.
E(ER)	Index	6n	E plus the contents of the register specified, ER, is the address of the operand.
@E(ER)	Deferred Index	7n	E added to ER gives the pointer to the address of the operand.
#E	Immediate	27	E is the operand.
@#E	Absolute	37	E is the address of the operand.
E	Relative	67	E is the address of the operand.
@E	Deferred Relative	77	E is the pointer to the address of the operand.

B.3 INSTRUCTIONS

The instructions which follow are grouped according to the operands they take and the bit patterns of their op-codes.

In the representation of op-codes, the following symbols are used:

SS	Source operand specified by a 6-bit address mode.
DD	Destination operand specified by a 6-bit address mode.
XX	8-bit offset to a location (branch instructions)
R	Integer between 0 and 7 representing a general register.

Symbols used in the description of instruction operations are:

SE	Source Effective address
DE	Destination Effective address
( )	Contents of
→	Becomes

The condition codes in the processor status word (PS) are affected by the instructions. These condition codes are represented as follows:

N	Negative bit:	set if the result is negative
Z	Zero bit:	set if the result is zero
V	oVerflow bit:	set if the operation caused an overflow
C	Carry bit:	set if the operation caused a carry

In the representation of the instruction's effect on the condition codes, the following symbols are used:

*	Conditionally set
-	Not affected
0	Cleared
1	Set

To set conditionally means to use the instruction's result to determine the state of the code (see the PDP-11 Handbook).

Logical operations are represented by the following symbols:

!	Inclusive OR
⊕	Exclusive OR
&	AND
¯	(used over a symbol) NOT (i.e., 1's complement)

B.3.1 Double-Operand Instructions Op A,A

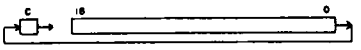
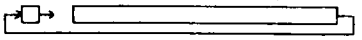
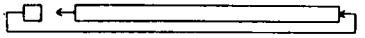
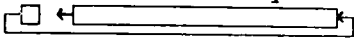
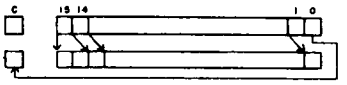
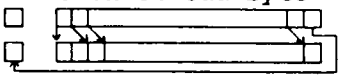
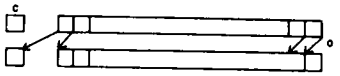
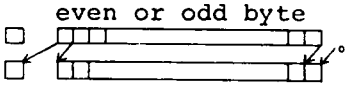
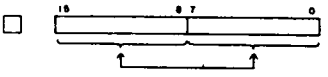
Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
01SSDD	MOV	MOVe	(SE)→ DE	*	*	0	-
11SSDD	MOVB	MOVe Byte					
02SSDD	CMP	CoMPare	(SE)-(DE)	*	*	*	*
12SSDD	CMPB	CoMPare Byte					
03SSDD	BIT	BIt Test	(SE)&(DE)	*	*	0	-
13SSDD	BITB	BIt Test Byte					
04SSDD	BIC	BIt Clear	($\overline{SE}$)&(DE)→ DE	*	*	0	-
14SSDD	BICB	BIt Clear Byte					
05SSDD	BIS	BIt Set	(SE)!(DE)→ DE	*	*	0	-
15SSDD	BISB	BIt Set Byte					
06SSDD	ADD	ADD	(SE)+(DE)→ DE	*	*	*	*
16SSDD	SUB	SUBtract	(DE)-(SE)→ DE	*	*	*	*

B.3.2 Single-Operand Instructions Op A

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
0050DD	CLR	CLeaR	Ø→ DE	0	1	0	0
1050DD	CLRB	CLeaR Byte					
0051DD	COM	COMplement	($\overline{DE}$)→ DE	*	*	0	1
1051DD	COMB	COMplement Byte					
0052DD	INC	INCrement	(DE)+1→ DE	*	*	*	-
1052DD	INCB	INCrement Byte					
0053DD	DEC	DECrement	(DE)-1→ DE	*	*	*	-
1053DD	DECB	DECrement Byte					
0054DD	NEG	NEGate	($\overline{DE}$)+1→ DE	*	*	*	*
1054DD	NEGB	NEGate Byte					

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
0055DD	ADC	ADd Carry	(DE)+(C)→ DE	*	*	*	*
1055DD	ADCB	ADd Carry Byte					
0056DD	SBC	SuBtract Carry	(DE)-(C)→ DE	*	*	*	*
1056DD	SBCB	SuBtract Carry Byte					
0057DD	TST	TeST	(DE)-Ø→ DE	*	*	0	0
1057DD	TSTB	TeST Byte					

B.3.3 Rotate/Shift Instructions Op A

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
0060DD	ROR	ROtate Right		*	*	*	*
1060DD	RORB	ROtate Right Byte	even or odd byte 	*	*	*	*
0061DD	ROL	ROtate Left		*	*	*	*
1061DD	ROLB	ROtate Left Byte	even or odd byte 	*	*	*	*
0062DD	ASR	Arithmetic Shift Right		*	*	*	*
1062DD	ASRB	Arithmetic Shift Right Byte	even or odd byte 	*	*	*	*
0063DD	ASL	Arithmetic Shift Left		*	*	*	*
1063DD	ASLB	Arithmetic Shift Left Byte	even or odd byte 	*	*	*	*
0001DD	JMP	JuMP	DE→ PC	-	-	-	-
0003DD	SWAB	SWAp Bytes		*	*	0	0

B.3.4 Operate Instructions

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
000000	HALT	HALT	The computer stops all functions.	-	-	-	-
000001	WAIT	WAIT	The computer stops and and waits for an interrupt.	-	-	-	-
000002	RTI	ReTurn from Interrupt	The PC and PS are popped off the SP stack: ((SP))→ PC (SP)+2→ SP ((SP))→ PS (SP)+2→ SP RTI is also used to return from a trap.	*	*	*	*
000005	RESET	RESET	Returns all I/O devices to power-on state.	-	-	-	-

Condition Code Operates

Op-Code	Mnemonic	Stands for
000241	CLC	CLear Carry bit in PS.
000261	SEC	SEt Carry bit.
000242	CLV	CLear oVerflow bit.
000262	SEV	SEt oVerflow bit.
000244	CLZ	CLear Zero bit.
000264	SEZ	SEt Zero bit.
000250	CLN	CLear Negative bit.
000270	SEN	SEt Negative bit.
000254	CNZ	CLear Negative and Zero bits.
000257	CCC	Clear all Condition Codes
000277	SCC	Set all Condition Codes.
000240	NOP	No OPeration.

B.3.5 Trap Instructions Op or Op E where $0 \leq E \leq 377_8$
*OP (only)

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
*000003	(none)	(breakpoint trap)	Trap to location 14. This is used to call ODT.	*	*	*	*
*000004	IOT	Input/Output Trap	Trap to location 20. This is used to call IOX.	*	*	*	*
104000-104377	EMT	EMulator Trap	Trap to location 30. This is used to call system programs.	*	*	*	*
104400 104777	TRAP	TRAP	Trap to location 34. This is used to call any routine desired by the programmer.	*	*	*	*

B.3.6 Branch Instructions Op E where $-128_{10} \leq (E--2)/2 \leq 127_{10}$

Op-Code	Mnemonic	Stands for	Condition to be met if branch is to occur
0004XX	BR	BRanch always	
0010XX	BNE	Branch if Not Equal (to zero)	Z=0
0014XX	BEQ	Branch if EQual (to zero)	Z=1
0020XX	BGE	Branch if Greater than or Equal (to zero)	N(!)V=0
0024XX	BLT	Branch if Less Than (zero)	N (!) V=1
0030XX	BGT	Branch if Greater Than (zero)	Z!(N(!)V)=0
0034XX	BLE	Branch if Less than or Equal (to zero)	Z!(N(!)V)=1
1000XX	BPL	Branch if PLus	N=0
1004XX	BMI	Branch if MInus	N=1
1010XX	BHI	Branch if HIgher	C ! Z = 0
1014XX	BLOS	Branch if LOwer or Same	C ! Z = 1
1020XX	BVC	Branch if oVerflow Clear	V=0
1024XX	BVS	Branch if oVerflow Set	V=1
1030XX	BCC (or BHIS)	Branch if Carry Clear (or Branch if HIgher or Same)	C=0
1034XX	BCS (or BLO)	Branch if Carry Set (or Branch if LOwer)	C=1

B.3.7 Subroutine Call Op ER, A

Op-Code	Mnemonic	Stands for	Operation
004RDD	JSR	Jump to SubRoutine	Push register on the SP stack, put the PC in the register: DE→ TEMP - a temporary storage register internal to processor. (SP)-2→ SP (REG)→ (SP) (PC)→ REG (TEMP)→ PC

B.3.8 Subroutine Return Op ER

00020R	RTS	ReTurn from Subroutine	Put register in PC and pop old contents from SP stack into register.

B.4 ASSEMBLER DIRECTIVES

Mnemonic	Operand	Stands for	Operation
.EOT	none	End Of Tape	Indicates the physical end of the source input medium
.EVEN	none	EVEN	Insures that the assembly location counter is even by adding 1 if it is odd.
.END	E	END	Indicates the physical end of the program and optionally specifies the transfer address (E).
.WORD	E, E,... E, E,...	WORD (the void operator)	Generates words of data Generates words of data
.BYTE	E,E,...	BYTE	Generates bytes of data
.ASCII	/xxx...x/	ASCII	Generates 7-bit ASCII characters for text enclosed by delimiters.
.TITLE	name	TITLE	Generates a name for the object module.
.ASECT	none	Absolute SECTion	Initiates the Absolute section.
.CSECT	name	Control SECTion	Initiates and identifies Relocatable Program section (section is unnamed if no operand).
.LIMIT	none	LIMIT	Generates two words containing the low and high limits of the relocatable section.
.GLOBL	name,name,...	GLOBaL	Specifies each name to be a global symbol.
.RAD50	/XXX/	RADix 50	Generates the RADIX 50 representation of the ASCII characters in delimiters.

Mnemonic	Operand	Stands For	Operation
.IFZ or .IFEQ	E	IF E=0	Assemble what follows up to the terminating .ENDC if the expression E is 0.
.IFNZ or .IFNE	E	IF E≠0	Assemble what follows up to the terminating .ENDC, if the expression E is not 0.
.IFL or .IFLT	E	IF E<0	Assemble what follows up to the terminating .ENDC, if the expression E is less than 0.
.IFLE	E	IF E≤0	Assemble what follows up to the terminating .ENDC, if the expression E is less than or equal to 0.
.IFG or .IFGT	E	IF E>0	Assemble what follows up to the terminating .ENDC, if the expression E is greater than 0.
.IFGE	E	IF E≥0	Assemble what follows up to the terminating .ENDC, if the expression E is greater than or equal to 0.
.IFDF	name	IF name DeFined	Assemble what follows up to the terminating .ENDC if the symbol name is defined.
.IFNDF	name	IF name uNDeFined	Assemble what follows up to the terminating .ENDC if the symbol name is undefined.
.ENDC	none	END of Conditional	Terminates the range of a conditional directive.

B.5 ERROR CODES

Error Code	Meaning
A	Addressing error. An address within the instruction is incorrect. Also includes relocation errors.
B	Bounding error. Instructions or word data are being assembled at an odd address in memory.
D	Doubly-defined symbol referenced. Reference was made to a symbol which is defined more than once.
I	Illegal character detected. Illegal characters which are also non-printing are replaced by a ? on the listing.

Error Code	Meaning
L	Line buffer overflow, i.e., input line exceeds 83 characters. All extra characters beyond 72_{10} are ignored.
M	Multiple definition of a label. A label was encountered which was equivalent (in the first six characters) to a previously encountered label.
N	Number containing an 8 or 9 was not terminated by a decimal point.
P	Phase error. A label's definition or value varies from one pass to another.
Q	Questionable syntax. There are missing arguments or the instruction scan was not completed, or a carriage return was not followed by a linefeed or form feed.
R	Register-type error. An invalid use of or reference to a register has been made.
T	Truncation error. More than the allotted number of bits were input so the leftmost bits were truncated. T error does not occur for the result of an expression.
U	Undefined symbol. An undefined symbol was encountered during the evaluation of an expression. Relative to the expression, the undefined symbol is assigned a value of zero.

APPENDIX C

LINKING PAL-11R AND ITS OVERLAY BUILDER AND CONSTRUCTING THE RUN TIME SYSTEM

At assembly time, the Assembler exists as:

1. a load module of its core resident portion, and
2. N non-resident overlays in a contiguous file.

The Assembler package is delivered to the customer as three (3) object modules:

1. the Assembler (PAL11R)
2. the Permanent Symbol Table (PALSYM)
3. the Overlay Builder (PALOB)

Two links must be built from these object modules:

1. the Assembler and the Permanent Symbol Table
2. the Overlay Builder

It is imperative that both links have exactly the same TOP specified at link time.

The load module of the Overlay Builder must now be executed via the Monitor command RUN, while logged in under UIC 1,1. It will produce a contiguous file, PAL11R.OVR, on the system device[1], with User Identification Code [1,1]. Monitor commands CTRL/C BEGIN or CTRL/C RESTART will abort the build process and automatically begin it again. When the building is done, control will return to the Monitor.

The load module of the Assembler/Permanent Symbol Table may now be executed via the monitor command RUN. This run requires the presence of the overlay file, PAL11R.OVR, on the system device[1], with User Identification Code [1,1].

Overlay Size and Makeup

The size of each overlay on the overlay device and the size of the

[1]The file specifier and device may be changed with the Monitor command ASSIGN on logical dataset OVR prior to issuing the RUN command.

overlay area in core is currently 512 decimal words. This size was chosen after examination of the segmentable sections of the Assembler and their relationships to each other.

There are currently five overlays:

```
OVERLAY #1
   Assembler Initialization
   CSI Interface
   CTRL/C RESTART handler
OVERLAY #2
   .CSECT   assembler directive
   .ASECT      "          "
   .GLOBL      "          "
   .TITLE      "          "
   .LIMIT      "          "
   GSD (Global Symbol Directory) Output
   Linear Search Routine (used by .CSECT and
            GSD Output routine)
OVERLAY #3
   .END  assembler directive
   Conditional assembler directives
   Symbol Table Lister
OVERLAY #4
   .ASCII  assembler directive
   .BYTE       "          "
   .RAD5Ø      "          "
   .WORD       "          "
   Direct Assignment processor
   PASS Initialization
OVERLAY #5
   GET INPUT LINE ROUTINE
   STATEMENT EVALUATOR
   LABEL FIELD PROCESSOR
   INSTRUCTION PROCESSORS
   PHYSICAL END OF INPUT (EOD) PROCESSOR
```

Programming Hints for the User

The user can organize his source code to cut down on the number of overlay transfers needed to accomplish his assembly.

He should gather all his executable code into one area because once OVERLAY #5 is loaded into core to process source lines restricted to labels and instructions it will remain in core until one of the following occurs:

1. an assembly directive is encountered

   e.g., .WORD
   .CSECT

2. a non-instruction line defaults to .WORD

   e.g., ADDR

3. a direct assignment statement is encountered

   e.g., A=1ØØ
   .=.+6Ø

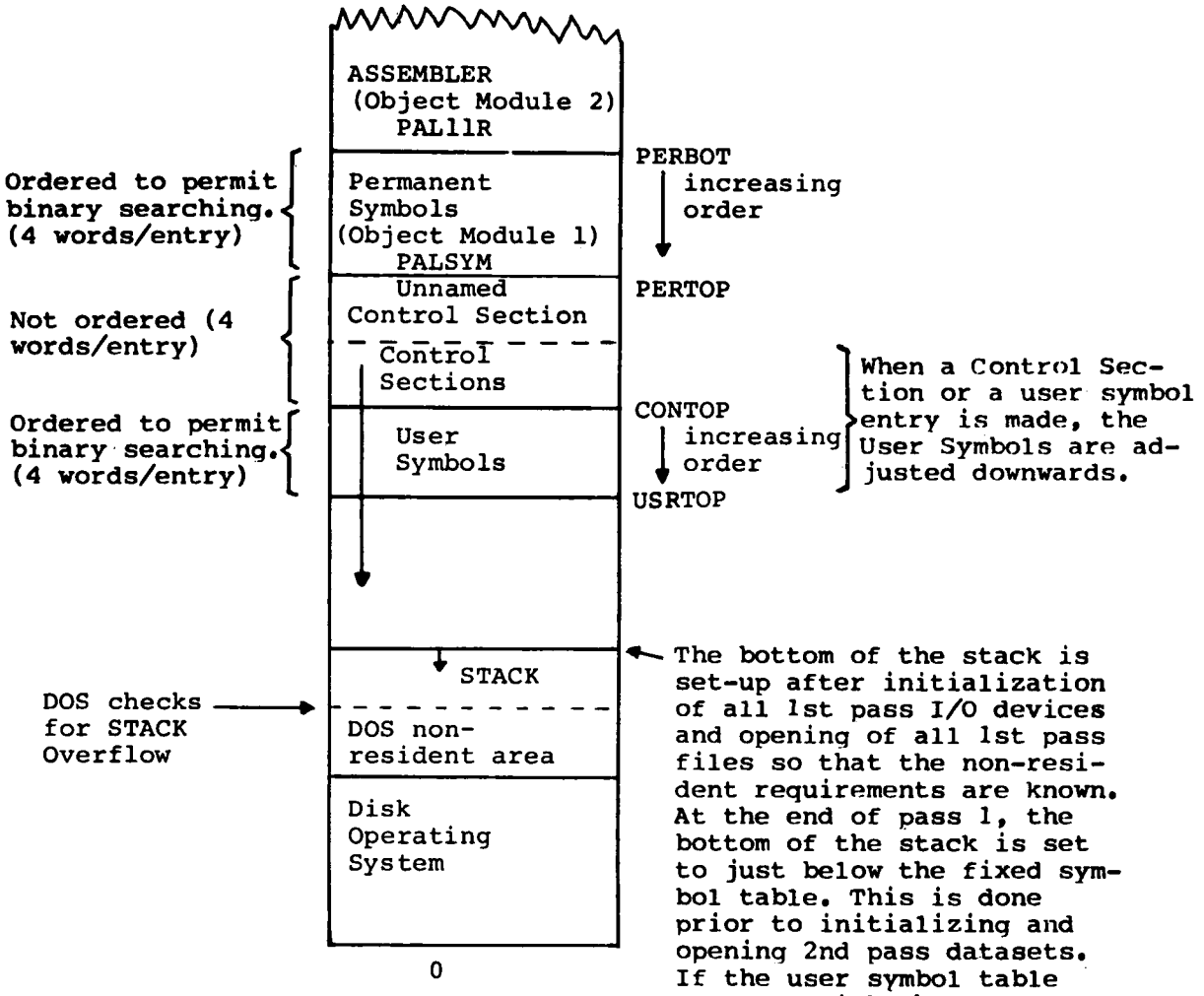
4. the end of a pass of the assembly process

Linking the Assembler and the Permanent Symbol Table

The load module of the Assembler and the Permanent Symbol Table should be given the file name:

PAL11R

PALSYM must be the first object module input to the Linker and PAL11R the second. This order allows for assembly time core utilization as illustrated by the memory map on the following page.


ASSEMBLER
(Object Module 2)
PAL11R
Ordered to permit binary searching. (4 words/entry)
Permanent Symbols
(Object Module 1)
PALSYM
PERBOT
increasing order
Not ordered (4 words/entry)
Unnamed Control Section
PERTOP
Control Sections
Ordered to permit binary searching. (4 words/entry)
User Symbols
CONTOP
increasing order
USRTOP
When a Control Section or a user symbol entry is made, the User Symbols are adjusted downwards.
STACK
DOS checks for STACK Overflow
DOS non-resident area
Disk Operating System
0
The bottom of the stack is set-up after initialization of all 1st pass I/O devices and opening of all 1st pass files so that the non-resident requirements are known. At the end of pass 1, the bottom of the stack is set to just below the fixed symbol table. This is done prior to initializing and opening 2nd pass datasets. If the user symbol table collides with the bottom of the stack, the stack is moved down in core in an attempt to continue the assembly. This may result in a STACK overflow condition.

The format of the Permanent Symbol Table (PST) is described below to assist users who wish to add and/or delete symbols.

The PST is a separate object module which is loaded to reside immediately below the assembler in core memory. Once assembled, it is fixed in size; however, it may be edited, assembled, and linked to suit the specific needs of customers.

The PST is bounded by the internal global symbols SYMTBB and SYMTBT, where the former is the highest address of the PST and the latter is the address of the 1st word below the PST.

The PST is ordered (to permit binary searching) with the smallest symbol (in Radix50 packed notation) high in core and the largest symbol (in Radix50 packed notation) low in core.

Radix50 character representation:

```
A thru Z  :  1 thru 32 respectively
       $  :  33
       .  :  34
0 thru 9  :  36 thru 47 respectively
```

Radix50 packed notation:

```
1st packed triad = CHAR1*50*50+CHAR2*50+CHAR3
2nd packed triad = CHAR4*50*50+CHAR5*50+CHAR6
```

Each entry is 4 words with the lowest word containing the 1st packed triad. The next lowest containing the 2nd packed triad. The next lowest containing the value (which for assembler directives is an external global to be linked to the appropriate processor in PAL-11R) and the highest word containing the flags in the low byte and the control section ID (which is always 0 for PST entries) in the high byte.

Internal global symbols:

```
       .GLOBL SYMTBB, SYMTBT
```

External global symbols:

```
       .GLOBL  ASCII, ASECT, BYTE, CSECT, END-
       .GLOBT  ENDC, EOT, EVEN, GLOBL, IFDF
       .GLOBL  IFG, IFGE, IFL, IFLE, IFNDF
       .GLOBL  IFNZ, IFZ, LIMIT, RAD50
       .GLOBL  TITLE, WORD
```

Flags:

ASMDIR=10	Bit 3 being on in the flag Byte indicates that this PST entry is an assembler directive.
BYTFLG=1	Bit 0 being on in the flag Byte indicates that this PST entry is byte enabled. This allows one entry to satisfy searches for word and byte instructions. E.G. the entry 'MOV', because bit 0 is on, will satisfy searches for 'MOV' or 'MOVB'

Instruction Class:
Bits 4-7 of the flag byte designate the type of instruction to provide dispatch information to PAL-11R.

SCLAS0=0	Operate group
SCLAS1=20	Unary group
SCLAS2=40	Binary group
SCLAS3=60	RTS
SCLAS4=100	Branch group
SCLAS5=120	JSR
SCLAS6=140	TRAP group

APPENDIX D

PERMANENT SYMBOL TABLE

A listing of the Permanent Symbol Table follows:

```
        .TITLE PALSYM
;THE PERMANENT SYMBOL TABLE (PST)
;
;       VERSION LEVEL 2
;       PATCH LEVEL A
;
;       COPYRIGHT 1971, DIGITAL EQUIPMENT CORP.
;       MAYNARD, MASSACHUSETTS
;
        ;THE PST IS A SEPERATE OBJECT MODULE WHICH
        ;IS LOADED TO RESIDE IMMEDIATELY BELOW THE
        ;ASSEMBLER IN CORE MEMORY.  ONCE ASSEMBLED, IT
        ;IS FIXED IN SIZE; HOWEVER, IT MAY BE
        ;EDITED, ASSEMBLED, AND LINKED TO SUIT THE
        ;SPECIFIC NEEDS OF CUSTOMERS.
        ;THE PST IS BOUNDED BY THE INTERNAL GLOBAL
        ;SYMBOLS SYMTBB AND SYMTBT, WHERE THE
        ;FORMER IS THE HIGHEST ADDRESS OF THE
        ;PST AND THE LATTER IS THE ADDRESS OF
        ;THE 1ST WORD BELOW THE PST.
        ;THE PST IS ORDERED (TO PERMIT BINARY
        ;SEARCHING) WITH THE SMALLEST SYMBOL (IN
        ;MOD40 PACKED NOTATION) HIGH IN CORE AND
        ;THE LARGEST SYMBOL (IN MOD40 PACKED NOTATION)
        ;LOW IN CORE.
        ;MOD40 CHARACTER REPRESENTATION
        ;A THRU Z  : 1 THRU 26 RESPECTIVELY
        ;       $  :  27
        ;       .  :  28
        ;0 THRU 9  : 30 THRU 39 RESPECTIVELY
        ;MOD40 PACKED NOTATION
        ;1ST PACKED TRIAD = CHAR1*40*40+CHAR2*40+CHAR3
        ;2ND PACKED TRIAD = CHAR4*40*40+CHAR5*40+CHAR6
        ;
        ;EACH ENTRY IS 4 WORDS WITH THE LOWEST WORD
        ;CONTAINING THE 1ST PACKED TRIAD, THE NEXT
        ;LOWEST CONTAINING THE 2ND PACKED TRIAD, THE
        ;NEXT LOWEST CONTAINING THE VALUE (WHICH FOR
        ;ASSEMBLER DIRECTIVES IS AN EXTERNAL GLOBAL
        ;TO BE LINKED TO THE APPROPRIATE PROCESSOR IN
        ;PAL-11R) AND THE HIGHEST WORD CONTAINING
        ;THE FLAGS IN THE LOW BYTE AND THE CONTROL
        ;SECTION ID (WHICH IS ALWAYS 0 FOR PST
        ;ENTRIES) IN THE HIGH BYTE.
        ;
        ;INTERNAL GLOBAL SYMBOLS:
                .GLOBL SYMTBB, SYMTBT
        ;
        ;EXTERNAL GLOBAL SYMBOLS:
                .GLOBL  ASCII, ASECT, BYTE, CSECT, END
                .GLOBL  ENDC, EOT, EVEN, GLOBL, IFDF
                .GLOBL  IFG, IFGE, IFL, IFLE, IFNDF
                .GLOBL  IFNZ, IFZ, LIMIT, RAD50
                .GLOBL  TITLE, WORD
```

```
;
;FLAGS:
  ASMDIR=10       ;BIT 3 BEING ON IN THE FLAG
                  ;BYTE INDICATES THAT THIS PST
                  ;ENTRY IS AN ASSEMBLER DIRECTIVE.
  BYTFLG=1        ;BIT 0 BEING ON IN THE FLAG
                  ;BYTE INDICATES THAT THIS PST
                  ;ENTRY IS BYTE ENABLED.  THIS
                  ;ALLOWS ONE ENTRY TO SATISIFY
                  ;SEARCHES FOR WORD AND BYTE
                  ;INSTRUCTIONS.  E.G. THE ENTRY
                  ;!MOV!, BECAUSE BIT 0 IS ON,
                  ;WILL SATISFY SEARCHES FOR
                  ;!MOV! OR !MOVB!
;
;INSTRUCTION CLASS:
;BITS 4-7 OF THE FLAG BYTE DESIGNATE THE
;TYPE OF INSTRUCTION TO PROVIDE DISPATCH
;INFORMATION TO PAL-11R.
SCLAS0=0                  ;OPERATE GROUP
SCLAS1=20                 ;UNARY GROUP
SCLAS2=40                 ;BINARY GROUP
SCLAS3=60                 ;RTS
SCLAS4=100                ;BRANCH GROUP
SCLAS5=120                ;JSR
SCLAS6=140                ;TRAP GROUP
;
;
;
        .EVEN
SYMTBT: .WORD   0                 ;1ST REGISTER BELOW PST.
;
        .WORD   131247            ; .WORD
        .WORD   070440
        .WORD   WORD
        .WORD   ASMDIR
;
        .WORD   131051            ; .TITLE
        .WORD   077345
        .WORD   T-ITLE
        .WORD   ASMDIR
;
        .WORD   130721            ; .RAD50
        .WORD   017226
        .WORD   RAD50
        .WORD   ASMDIR
;
        .WORD   130351            ; .LIMIT
        .WORD   051274
        .WORD   LIMIT
        .WORD   ASMDIR
;
        .WORD   130156            ; .IFZ
        .WORD   121200
        .WORD   IFZ
        .WORD   ASMDIR
```

```
;
        .WORD   130156          ; .IFNZ
        .WORD   055620
        .WORD   IFNZ
        .WORD   ASMDIR
;
        .RAD50  /.IF/           ; .IFNE
        .RAD50  /NE/
        .WORD   IFNZ
        .WORD   ASMDIR
;
        .WORD   130155          ; .IFNDF
        .WORD   054046
        .WORD   IFNDF
        .WORD   ASMDIR
;
        .RAD50  /.IF/           ; .IFLT
        .RAD50  /LT/
        .WORD   IFL
        .WORD   ASMDIR
;
        .WORD   130156          ; .IFLE
        .WORD   045710
        .WORD   IFLE
        .WORD   ASMDIR
;
        .WORD   130156          ; .IFL
        .WORD   045400
        .WORD   IFL
        .WORD   ASMDIR
;
        .RAD50  /.IF/           ; .IFGT
        .RAD50  /GT/
        .WORD   IFG
        .WORD   ASMDIR
;
        .WORD   130156          ; .IFGE
        .WORD   026210
        .WORD   IFGE
        .WORD   ASMDIR
;
        .WORD   130156          ; .IFG
        .WORD   025700
        .WORD   IFG
        .WORD   ASMDIR
;
        .RAD50  /.IF/           ; .IFEQ
        .RAD50  /EQ/
        .WORD   IFZ
        .WORD   ASMDIR
;
        .WORD   130156          ; .IFDF
        .WORD   014760
        .WORD   IFDF
        .WORD   ASMDIR
;
        .WORD   130044          ; .GLOBL
        .WORD   057034
        .WORD   GLOBL
        .WORD   ASMDIR
;
        .WORD   127736          ; .EVEN
```

```
        .WORD   020560
        .WORD   EVEN
        .WORD   ASMDIR


;
        .WORD   127727          ; .EOT
        .WORD   076400
        .WORD   EOT
        .WORD   ASMDIR
;
        .WORD   127726          ; .ENDC
        .WORD   014570
        .WORD   ENDC
        .WORD   ASMDIR
;
        .WORD   127726          ; .END
        .WORD   014400
        .WORD   END
        .WORD   ASMDIR
;
        .WORD   127613          ; .CSECT
        .WORD   017714
        .WORD   CSECT
        .WORD   ASMDIR
;
        .WORD   127551          ; .BYTE
        .WORD   076710
        .WORD   BYTE
        .WORD   ASMDIR
;
        .WORD   127473          ; .ASECT
        .WORD   017714
        .WORD   ASECT
        .WORD   ASMDIR
;
        .WORD   127473          ; .ASCII
        .WORD   012061
        .WORD   ASCII
        .WORD   ASMDIR
;
;
;
        .WORD   107761          ;WAIT
        .WORD   076400
        .WORD   000001
        .WORD   SCLAS0
;
        .WORD   100014          ;TST
        .WORD   0
        .WORD   005700
        .WORD   SCLAS1+BYTFLG
;
        .WORD   077721          ;TRAP
        .WORD   062000
        .WORD   104400
        .WORD   SCLAS6
```

```
;
        .WORD   075131          ;SWAB
        .WORD   006200
        .WORD   000300
        .WORD   SCLAS1
;
        .WORD   075012          ;SUB
        .WORD   0
        .WORD   160000
        .WORD   SCLAS2
;
        .WORD   073642          ;SEZ
        .WORD   0
        .WORD   000264
        .WORD   SCLAS0
;
        .WORD   073636          ;SEV
        .WORD   0
        .WORD   000262
        .WORD   SCLAS0
;
        .WORD   073626          ;SEN
        .WORD   0
        .WORD   000270
        .WORD   SCLAS0
;
        .WORD   073613          ;SEC
        .WORD   0
        .WORD   000261
        .WORD   SCLAS0
;
        .WORD   073473          ;SCC
        .WORD   0
        .WORD   000277
        .WORD   SCLAS0
;
        .WORD   073423          ;SBC
        .WORD   0
        .WORD   005600
        .WORD   SCLAS1+BYTFLG
;
        .WORD   071663          ;RTS
        .WORD   0
        .WORD   000200
        .WORD   SCLAS3
;
        .WORD   071651          ;RTI
        .WORD   0
        .WORD   000002
        .WORD   SCLAS0
```

```
;
        .WORD   071352          ;ROR
        .WORD   0
        .WORD   006000
        .WORD   SCLAS1+BYTFLG
;
        .WORD   071344          ;ROL
        .WORD   0
        .WORD   006100
        .WORD   SCLAS1+BYTFLG
;
        .WORD   070533          ;RESET
        .WORD   021140
        .WORD   000005
        .WORD   SCLAS0
;
        .WORD   054750          ;NOP
        .WORD   0
        .WORD   000240
        .WORD   SCLAS0
;
        .WORD   054117          ;NEG
        .WORD   0
        .WORD   005400
        .WORD   SCLAS1+BYTFLG
;
        .WORD   051656          ;MOV
        .WORD   0
        .WORD   010000
        .WORD   SCLAS2+BYTFLG
;
        .WORD   040612          ;JSR
        .WORD   0
        .WORD   004000
        .WORD   SCLAS5
;
        .WORD   040230          ;JMP
        .WORD   0
        .WORD   000100
        .WORD   SCLAS1
;
        .WORD   035254          ;IOT
        .WORD   0
        .WORD   000004
        .WORD   SCLAS0
;
        .WORD   035163          ;INC
        .WORD   0
        .WORD   005200
        .WORD   SCLAS1+BYTFLG
```

```
/
        .WORD   031064          /HALT
        .WORD   076400
        .WORD   0
        .WORD   SCLAS0
/
        .WORD   020534          /EMT
        .WORD   0
        .WORD   104000
        .WORD   SCLAS6
/
        .WORD   014713          /DEC
        .WORD   0
        .WORD   005300
        .WORD   SCLAS1+BYTFLG
/
        .WORD   012445          /COM
        .WORD   0
        .WORD   005100
        .WORD   SCLAS1+BYTFLG
/
        .WORD   012412          /CNZ
        .WORD   0
        .WORD   000254
        .WORD   SCLAS0
/
        .WORD   012330          /CMP
        .WORD   0
        .WORD   020000
        .WORD   SCLAS2+BYTFLG
/
        .WORD   012272          /CLZ
        .WORD   0
        .WORD   000244
        .WORD   SCLAS0
/
        .WORD   012266          /CLV
        .WORD   0
        .WORD   000242
        .WORD   SCLAS0
/
        .WORD   012262          /CLR
        .WORD   0
        .WORD   005000
        .WORD   SCLAS1+BYTFLG
/
        .WORD   012256          /CLN
        .WORD   0
        .WORD   000250
        .WORD   SCLAS0
/
        .WORD   012243          /CLC
        .WORD   0
        .WORD   000241
        .WORD   SCLAS0
```

```
;
        .WORD   011473          ;CCC
        .WORD   0
        .WORD   000257
        .WORD   SCLAS0
;
        .WORD   010003          ;BVS
        .WORD   0
        .WORD   102400
        .WORD   SCLAS4
;
        .WORD   007763          ;BVC
        .WORD   0
        .WORD   102000
        .WORD   SCLAS4
;
        .WORD   007520          ;BR
        .WORD   0
        .WORD   000400
        .WORD   SCLAS4
;
        .WORD   007414          ;BPL
        .WORD   0
        .WORD   100000
        .WORD   SCLAS4
;
        .WORD   007265          ;BNE
        .WORD   0
        .WORD   001000
        .WORD   SCLAS4
;
        .WORD   007221          ;BMI
        .WORD   0
        .WORD   100400
        .WORD   SCLAS4
;
        .WORD   007164          ;BLT
        .WORD   0
        .WORD   002400
        .WORD   SCLAS4
;
        .WORD   007157          ;BLOS
        .WORD   073300
        .WORD   101400
        .WORD   SCLAS4
;
        .WORD   007157          ;BLO
        .WORD   0
        .WORD   103400
        .WORD   SCLAS4
```

```
;
        .WORD   007145          ;BLE
        .WORD   0
        .WORD   003400
        .WORD   SCLAS4
;
        .WORD   006774          ;BIT
        .WORD   0
        .WORD   030000
        .WORD   SCLAS2+BYTFLG
;
        .WORD   006773          ;BIS
        .WORD   0
        .WORD   050000
        .WORD   SCLAS2+BYTFLG
;
        .WORD   006753          ;BIC
        .WORD   0
        .WORD   040000
        .WORD   SCLAS2+BYTFLG
;
        .WORD   006711          ;BHIS
        .WORD   073300
        .WORD   103000
        .WORD   SCLAS4
;
        .WORD   006711          ;BHI
        .WORD   0
        .WORD   101000
        .WORD   SCLAS4
;
        .WORD   006654          ;BGT
        .WORD   0
        .WORD   003000
        .WORD   SCLAS4
;
        .WORD   006635          ;BGE
        .WORD   0
        .WORD   002000
        .WORD   SCLAS4
;
        .WORD   006531          ;BEQ
        .WORD   0
        .WORD   001400
        .WORD   SCLAS4
```

```
;
        .WORD   006413          ;BCS
        .WORD   0
        .WORD   103422
        .WORD   SCLAS4
;
        .WORD   006373          ;BCC
        .WORD   0
        .WORD   103000
        .WORD   SCLAS4
;
        .WORD   004512          ;ASR
        .WORD   0
        .WORD   006200
        .WORD   SCLAS1+BYTFLG
;
        .WORD   004504          ;ASL
        .WORD   0
        .WORD   006300
        .WORD   SCLAS1+BYTFLG
;
        .WORD   003344          ;ADD
        .WORD   0
        .WORD   060000
        .WORD   SCLAS2
;
        .WORD   003343          ;ADC
        .WORD   0
        .WORD   005500
        .WORD   SCLAS1+BYTFLG
;
;
;
SYMTBR=.-2                      ;HIGHEST ADDRESS OF PST.

        .END
```

APPENDIX E

SYSTEM ERROR MESSAGES

		Error Code
1.	**Command String Semantic Errors**	
1.1	Illegal switch Too many switches Illegal switch value Too many switch values	S2Ø3
1.2	Too many output file specifications	S2Ø4
1.3	Too many input file specifications	S2Ø5
1.4	Input file missing	S2Ø6
2.	**Limit Errors**	
2.1	More than 376_8 .CSECT's	S2ØØ
2.2	More than 177_8 nested conditionals	S2Ø1
3.	**I/O Errors**	
3.1	Binary or listing file structured device full or input command string too long	S2Ø2
3.2	Input .TRAN error on Assembler's overlay file	S2Ø7

APPENDIX F

VØØ5A Programming Notes

F.1 Notes on Size Requirements

The symbol table capacity of PAL-11R VØØ5A running under the Disk Operating System and utilizing overlays varies as a function of the I/O devices involved in the assembly. This is extremely relevant for 8K of core memory configurations.

At assembly time, 5497 decimal words are always occupied by the following items:

Resident PAL-11R	32Ø9
Permanent Symbol Table	336
PAL-11R's stack	1ØØ*
Absolute loader	98
Unnamed .CSECT entry	4
User Current Location "." symbol	4
Resident DOS (RF11 Disk)	1746**
	5497

In addition to this figure, consider the non-resident requirements of DOS which consist of the buffers and drivers needed for the assembly and 256_{10} words for the .OPEN processor.

The Assembler allows for:

two input devices (files)

three output devices (files)

i.e., binary
listing
symbol table

and the command output device.

The command output device and buffer is always required and occupies a total of 56Ø decimal words.

```
5497
+56Ø
6Ø57
```

---

*The Assembler will decrease its stack size in favor of symbol table entries. If the symbol table requirements are still too large, the assembly will be aborted with the FØØ1 (STACK OVERFLOW) DOS error message.

**If the system device is RK11, this figure increases by 144_{10} words. If the system device is RC11, this figure decreases by 7 words.

Periodically during every pass of the assembly, the .OPEN processor will be needed; thus, for symbol table purposes it can be considered as a permanent requirement.

```
6Ø57
+256
6313
```

The input file or files is required for the duration of an assembly. The medium chosen for input directly affects symbol table storage space, e.g.,

	Buffer Size per File	Driver Size
RF11 or RC11 Disk	96 decimal words	system device included in resident DOS
RK11 Disk	288 decimal words	
DECtape	288 decimal words	165 decimal words

The user saves 357 decimal words if his input file resides on the RF or RC disk rather than on DECtape. Thus, PIPing the sources from DECtape to disk prior to assembly increases symbol table storage by 89 entries (4 words per symbol).

The output files are only required during their respective assembly passes; that is,

symbol table	-	end of pass 1
listing	-	pass 2
binary	-	pass 2

Choice of output medium has the same benefits here and, additionally, the user can "string out" the non-resident needs by specifying via the PASS command string option that the listing be output during pass 3.

Three examples, one worst case and the other two considerably better, will illustrate the above-mentioned facts:

EXAMPLE #1 (Worst case):

```
#DT1:TEST.OBJ,DT2:TEST.LST,DT2:TEST.SYM<DT3:TEST.PAL
      ↑           ↑           ↑           ↑
     288         288         288*      165+288
```

1Ø29+6313 = 7342 leaving room for 211 user symbols or .CSECT names in 8K of core.

*The binary file in pass 2 would use the same space required by the symbol table file at the end of pass 1.

EXAMPLE #2:

Note that when a device is not specified in the command string, the system device is taken as the default. Assuming the system device is RF or RC disk:

```
#TEST.OBJ,TEST.LST,TEST.SYM<TEST.PAL
  ↑         ↑         ↑        ↑
  96        96        96       96
```

288+6313 = 66Ø1 leaving room for 397 user symbols or .CSECT names in 8K of core.

The listing and symbol table files can later be PIPed from disk to a hard copy device such as the line printer, teleprinter, paper tape punch, etc.

EXAMPLE #3:

```
#TEST.OBJ,TEST.LST/PASS:3,TEST.SYM<TEST.PAL
  ↑         ↑                 ↑        ↑
  96        96                96*      96
```

The PASS:3 switch makes the binary and listing outputs occur during separate passes; thus, they share output buffers and effectively increase symbol table capacity by 96 words or 24 symbols. The total capacity is now 421 symbols.

The user should always be aware of his symbol table needs relative to what his hardware configuration can provide. If he does this, he can segment programs that will exceed the limits, assemble them separately, and link the object modules with Link-11.

F.2 Running the Overlay Builder (PALOB.LDA)

The overlay builder creates the file of overlays via logical data set name OVR which has the default file specifier:

sysdev:PAL11R.OVR[LOGIN UIC]<233>

It is recommended that the creation of the file of overlays be accomplished either while

1. logged in under [1,1]

*The binary file in pass 2 would use the same space required by the symbol table file at the end of pass 1.

or

2. with the Monitor ASSIGN command overriding the LOGIN UIC, e.g.:

```
.AS sysdev:PAL11R.OVR[1,1],OVR
```

In either of the above environments, the overlay builder can be executed, e.g.:

```
.RUN sysdev:PALOB.LDA*
```

and will create the file of overlays, PAL11R.OVR, on the system disk with a UIC of [1,1] and then will exit to the Monitor which is indicated by the output of the "$".

Transfer the Assembler from DECtape to the Monitor disk area with PIP.

```
$PAL[1,1]<DTØ:PAL11R.LDA**
```

F.3 Running the Assembler

The Assembler can now be executed via the Monitor command RUN:

```
.RUN    PAL
```

it will search for its overlays first under the file specifier

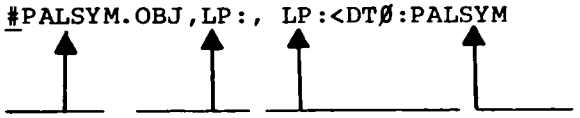
```
sysdev:PAL11R.OVR[LOGIN UIC]
```

and, if not found, then under

```
sysdev:PAL11R.OVR[1,1]
```

A typical command to PAL-11R to assemble the source of PALSYM that exists on DECtape with the file name PALSYM.PAL would be:

```
#PALSYM.OBJ,LP:, LP:<DTØ:PALSYM
```



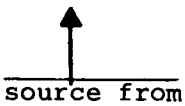
object module output	listing to Line Printer	Symbol Table to Line Printer	Source from DTØ

*A UIC specification is required if PALOB.LDA is under a UIC other than the LOGIN UIC.

**A UIC specification will be required if PAL11R.LDA is under a UIC other than the LOGIN UIC.

If the source was on paper tape, the command would read:

```
#PALSYM.OBJ,LP:,LP:<PR
```



The Monitor informs the user when to load the reader with the tape by typing

```
AØØ2    6332Ø
$
```

on the command output device. AØØ2 denotes device not ready; 6332Ø is PR (paper tape reader) in Radix-5Ø notation.

When the tape is ready, the user types the Monitor command COntinue.

The end of an assembly pass is indicated by END being printed on the command output device.

The end of an assembly is indicated by "#" (the command string request character) being printed on the command output device

F.4 Overlays on DECtape

Note that DECtape can be utilized as the overlay device. All that is required to accomplish this is:

1. Make the desired file specifier change prior to running PALOB.LDA

```
.AS DTØ:PALllR.OVR[1,1],OVR
.RUN DTØ:PALOB.LDA
```

2. Make the same file specifier change prior to running PAL

```
.AS DTØ:PALllR.OVR[1,1],OVR
.RUN PAL
```

F.5 Cautions

1. If no extension is specified for an input file, the Assembler will first search for a file using the default

extension "PAL". If this file is not present, the Assembler will then search for a file with a null extension. Both files being absent would result in a fatal Monitor error.

2. Macros for the DOS EMT's are not implemented. The user must type the expanded form.

3. @A(RØ)
   @A(RØ)+
   @A-(RØ)

   are all accepted by the Assembler as valid representations of the indexed deferred addressing mode @A(RØ).

F.6 Conditional Branch Instructions

When using the PDP-11 conditional branch instructions, it is imperative that the correct choice be made between the signed and the unsigned branches.

SIGNED	UNSIGNED
BGE	BHIS (BCC)
BLT	BLO
BGT	BHI
BLE	BLOS (BCS)

A common mistake is to use a signed branch (e.g., BGT) when comparing two memory addresses. This is incorrect when the addresses have opposite signs; that is, one address goes across the 16K (100000_8) boundary. This coding error usually appears as a result of relinking at different addresses and/or a change in program size.

APPENDIX G

FEATURES OF VØØ7A

Version VØØ7A of the PAL-11R Assembler contains a number of enhancements to provide a greater range of utility to the PDP-11 user. These enhancements can be broadly categorized as:

- PDP-11/45 support features; and,
- General improvements (multiply, divide in expressions).

G.1 PDP-11/45 SUPPORT FEATURES

Version VØØ7A provides a floating-point arithmetic capability, and instruction types to process the extended instruction set for the PDP-11/45 hardware.

G.1.1 Floating Point Arithmetic

The Floating-Point Processor option of the PDP-11/45 is complemented by the extended capabilities of version VØØ7A, which permits the coding of floating-point numbers and arguments in PAL-11R source programs.

G.1.1.1 Number Representation

A floating-point number is represented by a string of decimal digits. The string (which can be a single digit in length) may optionally contain a decimal point, and may be followed by an optional exponent indicator; in the form of the letter E and a signed decimal exponent. The list of number representations below contains seven distinct, valid representations of the same floating-point number:

```
3
3.
3.Ø
3.ØEØ
3EØ
.3E1
3ØØE-2
```

As can be quickly inferred, the list could be extended indefinitely (e.g., 3ØØØE-3, .Ø3E2, etc.). A leading plus sign is

ignored (e.g., +3.Ø is considered to be 3.Ø). Leading minus signs complement the sign bit. No other operators are allowed (e.g., 3.Ø+N is illegal).

Floating-point number representations are only valid in the contexts described in this appendix.

G.1.1.2 Floating-Point Directives

The assembly directives used for floating-point numbers are:

```
.FLT2
.FLT4
```

which generate two words, and four words, for floating-point arguments, respectively.

These directives are similar to the .WORD directive (see 8.7), but accept floating-point numbers as arguments.

EXAMPLES

```
1.  .FLT2 5E2,8E-2,25Ø
2.  .FLT4 16E-26,875E12
```

In Example 1, three 2-word arguments are specified (separated by commas). The PAL-11R Assembler stores each argument in a 2-word field, contiguous with the field of its neighboring argument. Thus if the source code were:

```
FLOAT: .FLT2 5E2,8E-2,25Ø
```

and FLOAT was defined as location 142Ø, the values 5ØØ, .Ø8, and 25Ø will be stored at locations 142Ø, 1424, and 143Ø, respectively.

The .FLT4 directive in Example 2 causes four words to be used to store the arguments. The actual format of the numbers is described in the PDP-11/45 Handbook.

The arguments included for these directives must each comprise one valid floating-point number.

In certain cases, 1-word floating-point numbers can be used as arguments: this is the result of using "immediate mode" addressing in the source field of floating-point source-double register type instructions (see G.1.2.8, below). Any numerical representation in the source field is converted to a 1-word floating-point number, rounded or truncated to 16 bits (depending on the mode in effect).

G.1.1.3 Rounding and Truncation

Floating-point numbers are normally rounded. That is, when a floating-point number exceeds the limits of the field in which it is to be stored, the high-order excess bit is added to the low-order retained bit. For example, if the number were to be stored in a 2-word field, but more than 32 bits were needed for its value, the highest bit carried out of the field would be added to the least significant position.

To change from rounding mode to truncation mode, specify the directive .TRUNC, as shown below.

```
.FLT2  12.34        ;ROUNDING MODE (DEFAULT)
.TRUNC              ;ENTER TRUNCATION MODE
.FLT2  12.34        ;TRUNCATE
.FLT2  12.34        ;CONTINUE TRUNCATION
.ROUND              ;RETURN TO ROUNDING
```

As indicated, the directive .ROUND is used to return from truncation to rounding mode.

G.1.2 Extended Instruction Set

This section describes the instructions that are included in the PDP-11/45, which are processed by version VØØ7A. These instructions are categorized by "types," as follows:

1. Source-Register;
2. Register-Destination;
3. Register-Offset;
4. Number;
5. Operate;
6. Destination;
7. Source;
8. Floating-Point Source-Double Register

9. Source-Double Register;
10. Double Register-Destination;
11. Shift Count-Register; and,
12. Priority.

Each instruction type is described, in terms of: its format; the mnemonic name of its associated instruction(s); and the octal op code for each instruction. Detailed descriptions of the instructions can be found in the PDP-11/45 Handbook.

The following notation conventions are observed in each instruction type description:

Abbreviation	Meaning
OC	Op code field in the binary word.
R	3-bit register field in the binary word; register symbol in source code.
DEST	Destination field.
SRC	Source field.
DR	Double register.
OPCODE	Instruction mnemonic.
PR	Priority level.

G.1.2.1 Source-Register Type

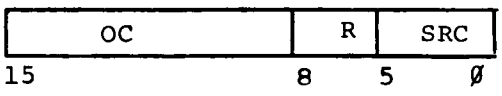
This instruction type has the following source code symbolic format:

```
OPCODE  SRC,R
```

Instructions included in this type are:

Mnemonic	Op Code	Function
DIV	071000	Divide
MUL	070000	Multiply

Binary Word Format



G.1.2.2 Register-Destination Type

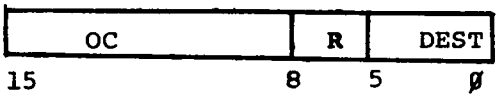
The symbolic format for this instruction type is:

```
OPCODE    R,DEST
```

The following instruction is included:

Mnemonic	Op Code	Function
XOR	074000	Exclusive OR

Binary Word Format



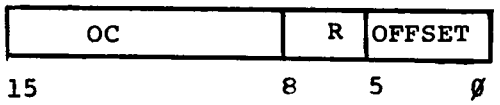
G.1.2.3 Register-Offset Type

Symbolic Format

```
OPCODE    R,ADDR
```

Mnemonic	Op Code	Function
SOB	077000	Subtract One and Branch

Binary Word Format



NOTE: The offset value is evaluated as

$$\frac{(PC - ADDR)}{2}$$

where PC = Program Counter.

The user must exercise caution in using the SOB instruction: Addressing errors will occur if an OFFSET value greater than 77_8 is generated; or if the address specified is a forward reference, relative to the SOB instruction.

G.1.2.4 Number Type

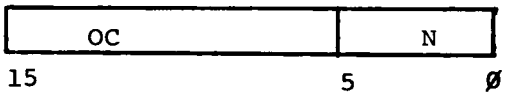
Symbolic Format

OPCODE N

"N" equals the number of arguments placed on the stack by the subroutine.

Mnemonic	Op Code	Function
MARK	006400	Stack clean-up on return from subroutine.

Binary Word Format



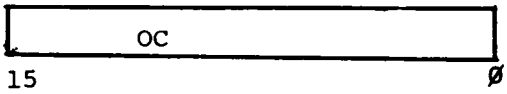
G.1.2.5 Operate Type

Symbolic Format

OPCODE

Mnemonic	Op Code	Function
BPT	000003	Breakpoint trap.
CFCC	170000	Copy Floating Condition Codes.
RTT	000006	Return from Interrupt.
SETD	170011	Set Double Floating Mode.
SETF	170001	Set Floating Mode.
SETI	170002	Set Integer Mode.
SETL	170012	Set Long Integer Mode.

Binary Word Format



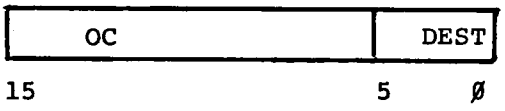
G.1.2.6 Destination Type

Symbolic Format

OPCODE DEST

Mnemonic	Op Code	Function
ABSD	170600	Make Absolute Double
ABSF	170600	Make Absolute Floating
CLRD	170400	Clear Double
CLRF	170400	Clear Floating
MTPD	106600	Move to Previous Data Space
MTPI	006600	Move to Previous Instruction Space
NEGD	170700	Negate Double
NEGF	170700	Negate Floating
STFPS	170200	Store FPP's Program Status
STST	170300	Store FPP's Status
SXT	006700	Sign Extend
TSTD	170500	Test Double
TSTF	170500	Test Floating

Binary Word Format



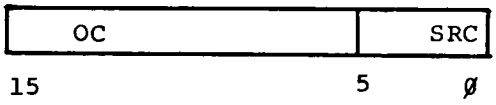
G.1.2.7 Source Type

Symbolic Format

OPCODE SRC

Mnemonic	Op Code	Function
LDFPS	170100	Load FPP's Program Status
MFPD	106500	Move from Previous Data Space
MFPI	006500	Move from Previous Instruction Space

Binary Word Format



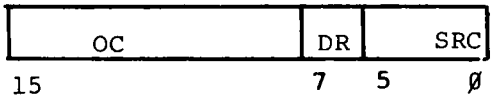
G.1.2.8 Floating-Point Source Double-Register Type

Symbolic Format

OPCODE SRC,DR

Mnemonics	Op Code	Function
ADDD	172000	Add Double
ADDF	172000	Add Floating
CMPD	173400	Compare Double
CMPF	173400	Compare Floating
DIVD	174400	Divide Double
DIVF	174400	Divide Floating
LDCDF	177400	Load and Convert, Double to Floating
LDCFD	177400	Load and Convert, Floating to Double
LDD	172400	Load Double
LDF	172400	Load Floating
MODD	171400	Multiply and Integerize Double
MODF	171400	Multiply and Integerize Floating
MULD	171000	Multiply Double
MULF	171000	Multiply Floating
SUBD	173000	Subtract Double
SUBF	173000	Subtract Floating

Binary Word Format



NOTE

When immediate-mode addressing is employed in the source field (SRC), any numeric representation in this field is assumed to be in floating-point notation (one word).

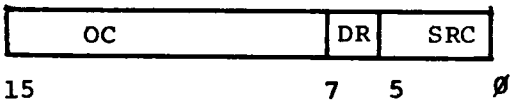
G.1.2.9 Source-Double Register Type

Symbolic Format

```
OPCODE    SRC,DR
```

Mnemonic	Op Code	Function
LDCID	177000	Load, Convert Integer to Double
LDCIF	177000	Load, Convert Integer to Floating
LDCLD	177000	Load, Convert Long Integer to Double
LDCLF	177000	Load, Convert Long Integer to Floating
LDEXP	176400	Load Exponent

Binary Word Format



G.1.2.10 Double Register-Destination Type

Symbolic Format

```
OPCODE    DR,DEST
```

Mnemonic	Op Code	Function
STCDF	176000	Store, Convert Double to Floating.
STCDI	175400	Store, Convert Double to Integer
STCDL	175400	Store, Convert Double to Long Integer
STCFD	176000	Store, Convert Floating to Double
STCFI	175400	Store, Convert Floating to Integer
STCFL	175400	Store, Convert Floating to Long Integer
STD	174000	Store Double
STEXP	175000	Store Exponent
STF	174000	Store Floating

Binary Word Format

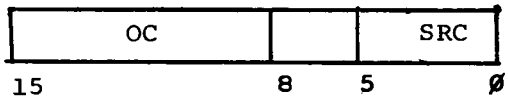
G.1.2.11 Shift Count-Register Type

Symbolic Format

OPCODE SRC,R

Instruction Mnemonic	Op Code	Function
ASH	072000	Arithmetic Shift
ASHC	073000	Arithmetic Shift Combined

Binary Word Format



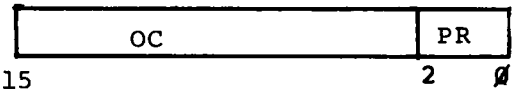
G.1.2.12 Priority Type

Symbolic Format

OPCODE PR

Mnemonic	Op Code	Function
SPL	000230	Set Priority Level

Binary Word Format



NOTE: PR is an integer from Ø to 7.

G.2 GENERAL IMPROVEMENTS

Version VØØ7A of the PAL-11R Assembler provides for the use of the multiply (*) and divide (/) operators, in addition to the operators described in Section 4.2 of the subject document.

Multiply and divide operations are performed in 2's complement integer arithmetic; in division, any remainder is discarded.

Expressions are evaluated from left to right. There is no operator hierarchy. Parenthetical groupings are not allowed.

The operands used in multiply and divide operations observe the same rules as those used in logical (AND, OR) operations; both operands must be absolute expressions. Legal and illegal expressions are illustrated below. In the examples, ABS is an absolute expression; REL is a relocatable expression; and EXT is an external .GLOBL expression.

LEGAL EXPRESSIONS	ILLEGAL EXPRESSIONS
.WORD ABS/ABS	.WORD REL/ABS
.WORD ABS*ABS	.WORD REL*ABS
	.WORD EXT/ABS
	.WORD EXT*ABS
	.WORD ABS/REL
	.WORD ABS*REL
	.WORD ABS/EXT

Although the above examples show the multiply and divide operators used in association with the assembler directive .WORD, they are by no means limited to such usage. These operators can be used in conjunction with any legal expression, such as:

```
.=.+ABS/2
  or
.BYTE  50/ABS
```

INDEX

Absolute symbol, see Symbols
Address modes, 7-2
Addressing, 7-1
ASCII conversion, 4-2, 8-7
Assembler directives, 8-1, 9-6
Assembly location counter, 2-2, 4-1, 5-1, 7-6, 7-7, 8-2, 8-5
Assembly passes, 5-1, 9-1, 9-6
Assignment, direct, 3-2, 9-8

Boolean operators, 4-2

Character set, 2-1
Command string, 9-1
  default condition, 9-2
  errors, 9-2
  format, 9-1
  switches, 9-3
Comment field, 2-4, 8-1

Datasets, 9-4
Decimal specification, 4-1
Direct assignment, 3-2, 5-1, 9-8
Dot, 2-2, 4-1, 5-1, 7-6, 7-7, 8-2, 8-5

Entry point symbol, see Symbols
Errors, 2-1, 2-2, 3-1, 3-4, 4-2, 8-6, 9-7
  codes for, 10-1
  command string, 9-2
  keyboard, 9-1
Expressions, 4-1
External symbol, see Symbols

Fields, statement, 2-1, 4-1, 5-1 7-7, 8-1, 8-5
Formatting, 2-4

Global symbols, see Symbols

Instruction formats, 7-7
Internal symbols, see Symbols

Labels, 2-2, 8-1, 8-3
Listing, 9-7
Load module, 1-1, 6-1, 8-5
Loading, 9-1
Location counter, 2-2, 4-1, 5-1, 7-6, 7-7, 8-2, 8-5

Logical operators, 4-2

Modes, address, 7-2

Numerical representation, 4-1

Object module, 1-1, 6-1, 8-1, 9-1, 9-8
Offsets,
  branch, 7-8
  relative mode, 7-5
Operand field (also see Address modes), 2-3, 4-1, 5-1, 7-7 8-1, 8-5, 8-10
Operating procedures, 9-1
Operator field, 2-3, 4-1, 8-1
Operators, arithmetic and logical, 4-2

Pages, 2-4
Paper tape reader, 9-6
Passes, assembly, 5-1, 9-1, 9-6
Permanent symbols, see Symbols
Program Counter (PC), 7-1, 7-4, 7-8
Pseudo-ops, see Assembler directives

Register symbols, see Symbols
Registers, general, 3-3, 7-1
Relocatable symbol, see Symbols
Restarting, 9-7

Starting, 9-1
Symbol tables, 3-1, 8-10, 9-1, 9-5
Symbols, 3-1, 6-1, 8-1
  absolute, 3-2, 4-3, 6-1, 7-8
  direct assignment, 3-2, 5-1, 9-8
  entry point, 3-2, 8-1
  external, 3-2, 4-3, 5-1, 6-1, 7-8, 8-1
  global, 1-1, 3-2, 4-1, 8-1, 9-8
  internal, 3-2, 8-4
  labels, 2-2, 8-1, 8-3
  permanent, 3-1, 4-1, 4-3
  register, 3-3
  relocatable, 3-2, 4-1, 4-3, 6-1, 7-8
  user-defined, 3-1, 9-1

Terminators, 2-1, 2-3, 8-10

DEC-11-EEDA-D

PDP-11

EDIT-11 TEXT EDITOR

PROGRAMMER'S MANUAL

for the

Disk Operating System

SOFTWARE SUPPORT CATEGORY

The software described in this document is supported by DEC under Category I, as defined on page iv of this document.

For additional copies, order No. DEC-11-EEDA-D from Digital Equipment Corporation, Direct Mail, Building 1-1, Maynard, Massachusetts 01754

First Printing, May 1971
Second Printing, October 1971

Your attention is invited to the last two pages of this document. The "How to Obtain Software Information" page tells you how to keep up-to-data with DEC's software. The "Reader's Comments" page, when filled in and mailed, is beneficial to both you and DEC; all comments received are acknoledged and are considered when documenting subsequent documents.

Copyright (C) 1971 by Digital Equipment Corporation

This document is for information purposes and is subject to change without notice.

Associated Documents:

PDP-11	Disk Operating System Monitor, Programmer's Handbook, DEC-11-MWDC-D
PDP-11	PAL-11R Assembler, Programmer's Manual, DEC-11-ASDC-D
PDP-11	ODT-11R Debugging Program, Programmer's Manual, DEC-11-OODA-D
PDP-11	Link-11 Linker and Libr-11 Librarian, Programmer's Manual, DEC-11-ZLDC-D
PDP-11	PIP, File Utility Package, Programmer's Manual, DEC-11-PIDB-D
PDP-11	FORTRAN IV Programmer's Manual, DEC-11-KFDA-D

Trademarks of Digital Equipemtn Corporation include:

DEC	PDP-11
Digital (logo)	Comtex-11
DECtape	RSTS-11
Unibus	RSX-11

P R E F A C E

This manual describes the features and operation of the Edit-11 Text Editor for the PDP-11 Disk Operating System (DOS). The manual assumes familiarity with the Disk Operating System Monitor (see PDP-11 Disk Operating System Monitor, Programmer's Handbook, DEC-11-MWDC-D).

In addition to the Edit-11 Text Editor and the DOS Monitor, the Disk Operating System software includes:

PAL-11R Assembler
ODT-11R Debugging Program
PIP-11, File Utility Package
Link-11 Linker
Libr-11 Librarian
FORTRAN IV

SOFTWARE SUPPORT CATEGORIES

Digital Equipment Corporation (DEC) makes available four categories of software. These categories reflect the types of support a customer may expect from DEC for a specified software product. DEC reserves the right to change the category of a software product at any time. The four categories are as follows:

CATEGORY I
Software Products Supported at no Charge

This classification includes current versions of monitors, programming languages, and support programs provided by DEC. DEC will provide installation (when applicable), advisory, and remedial support at no charge. These services are limited to original purchasers of DEC computer systems who have the requisite DEC equipment and software products.

At the option of DEC, a software product may be recategorized from Category I to Category II for a particular customer if the software product has been modified by the customer or a third party.

CATEGORY II
Software Products that Receive Support for a Fee

This category includes prior versions of Category I programs and all other programs available from DEC for which support is given. Programming assistance (additional support), as available, will be provided on these DEC programs and non-DEC programs when used in conjunction with these DEC programs and equipment supplied by DEC.

CATEGORY III
Pre-Release Software

DEC may elect to release certain software products to customers in order to facilitate final testing and/or customer familiarization. In this event, DEC will limit the use of such pre-release software to internal, non-competitive applications. Category III software is only supported by DEC where this support is consistent with evaluation of the software product. While DEC will be grateful for the reporting of any criticism and suggestions pertaining to a pre-release, there exists no commitment to respond to these reports.

CATEGORY IV
Non-Supported Software

This category includes all programs for which no support is given.

TABLE OF CONTENTS

		Page
CHAPTER 1	EDIT-11	1-1
CHAPTER 2	COMMAND MODE AND TEXT MODE	2-1
CHAPTER 3	COMMAND SYNTAX	3-1
3.1	The Character Location Pointer (Dot)	3-1
3.2	Mark	3-1
3.3	Character-Oriented Command Properties	3-1
3.4	Line-Oriented Command Properties	3-1
3.5	The Page Unit of Input	3-2
3.6	Arguments	3-2
3.7	Command Strings	3-3
3.8	Error Messages	3-4
CHAPTER 4	INPUT/OUTPUT	4-1
4.1	Primary Input and Output	4-1
4.2	Secondary Input and Output	4-1
CHAPTER 5	COMMANDS	5-1
5.1	Input and Output Commands	5-1
5.1.1	Read and Edit Read	5-1
5.1.2	List, Write and Edit Write	5-2
5.1.3	Next	5-2
5.1.4	Form Feed and Trailer	5-3
5.2	Editing Commands	5-3
5.2.1	Commands to Move Dot and Mark	5-3
5.2.1.1	Beginning	5-3
5.2.1.2	Jump and Advance	5-3
5.2.1.3	Mark	5-4
5.2.2	Search Commands	5-4
5.2.2.1	Get	5-4
5.2.2.2	wHole	5-4
5.2.2.3	Edit wHole	5-5
5.2.2.4	Position	5-5
5.2.2.5	Edit Position	5-6
5.2.3	Commands to Modify the Text	5-6
5.2.3.1	Insert	5-6
5.2.3.2	Delete and Kill	5-7

		Page
CHAPTER 5 (Cont'd)		
5.2.3.3	Change and eXchange	5-7
5.2.4	Utility Commands	5-9
5.2.4.1	Save	5-9
5.2.4.2	Unsave	5-9
5.2.4.3	Execute Macro	5-9
5.2.4.4	Edit Open	5-10
5.2.4.5	End File	5-10
5.2.4.6	EXit	5-10
CHAPTER 6	OPERATING INSTRUCTIONS	6-1
6.1	Starting	6-1
6.2	Restarting	6-2
6.3	Finishing an Edit	6-2
6.4	Error Recovery	6-2
6.5	Procedure with Low-Speed Punch	6-3
CHAPTER 7	IMPLEMENTATION NOTES	7-1
7.1	Macro Usage	7-1
7.2	Delimiter Usage	7-2
7.3	Subsidiary I/O	7-2
7.4	Save and Unsave	7-3
7.5	Creating a New File	7-3
7.6	Using EXit	7-3
Appendix A.	Error Diagnostics	A-1
Appendix B.	Commands	B-1

CHAPTER 1

EDIT-11

The PDP-11 Disk Editor (Edit-11) is a text editing program for use with the PDP-11 Disk Operating System. Operated by user commands from the keyboard, Edit-11 will read ASCII files from any device, make directed changes, and write on any device. In addition to basic editing functions, Edit-11 provides for command macros and multiple input and output files.

After initialization, Edit-11 indicates readiness to accept a command string by printing an asterisk (*). The user, through appropriate commands typed on the keyboard, causes Edit-11 to (1) read a "page" of text from the input file, (2) locate text in the buffer to be changed, (3) execute and verify changes, (4) output the page on the output file, and (5) proceed in the same manner to the end of his input file. After the last page of input is processed, the user closes his output file and returns to DOS Monitor control. The basic editing process above is quite flexible and made simple by Edit-11's large set of commands.

Each command to Edit-11 consists of one or two letters. Commands are typed in a string and sent to Edit-11 as a command string with appropriate arguments and text objects included. Edit-11 executes the command string, acting on the contents of its internal Page Buffer, in which the input source is stored.

The basic editing process can be divided into three sequential steps:

1. Reading of input text into a buffer internal to Edit-11.
2. Changing the text stored in the buffer.
3. Outputting the revised text to a new file.

CHAPTER 2

COMMAND MODE AND TEXT MODE

Whenever Edit-11 prints an * on the teleprinter, it is waiting to receive a command string, and is said to be in command mode. While most commands operate exclusively in command mode, there are eight commands that require additional text (the text object) to operate upon. Hence, although one can delete n characters by simply typing nD, the command to insert text, I, must be followed by the text to be inserted, the text object.

There are two ways to provide the required text object for a command. If the text object is small enough, Edit-11 can accept text in command mode if it is separated from the rest of the command string by delimiters. If the text object is long, or contains carriage return or line feed characters, then the user must cause Edit-11 to enter text mode.

Edit-11 processing begins in command mode. When you type a command string, no action occurs until you complete the string by typing the RETURN key (symbolized as <CR>). What happens next depends on the last character in the command string. If the last character in the string is a command requiring a text object, Edit-11 will enter text mode. If the last character is not a text command, Edit-11 expects to find any necessary test objects embedded in the command string itself, and will stay in command mode as it executes the command string.

Text can be accepted in command mode if the text contains no carriage return or line feed characters, and is small enough to fit on a single typed line with the command string. When Edit-11 finds a text command which is not the last command in the command string, it looks at the next character in the string as a text delimiter. All characters between this and the next occurrence of the delimiter are considered the text object for the command. Thus, any ASCII character which does not appear in the text object is a valid delimiter.

> The following example inserts OBJECT , then prints the entire buffer on the teleprinter (B/L). Note the delimiters (#) of the text object (OBJECT) for the Insert command (I).

```
*I#OBJECT# B/L
```

If the text object is lengthy or contains carriage return or line feed characters, it can only be accepted if Edit-11 is in text mode. To enter text mode, type the text command of interest as the last character in the command string. Then type the RETURN key. When Edit-11 executes this command, it will enter text mode, and accept text input. It continues to accept text until the user terminates text mode and reenters command mode by typing the LINE FEED key (symbolized as <LF>).

Note that typing the RETURN key always causes the physical return of the printing head to the beginning of the line, and automatically generates a line feed, thereby advancing the carriage to a new line. In text mode, the RETURN key serves these mechanical functions, allowing you to continue typing at the beginning of a new line, and at the same time enters a carriage return and line feed character into the text.

These are both counted as characters even though they do not print. When you wish to terminate text mode and reenter command mode, you must type the LINE FEED key. A typed line feed is not considered to be part of the text unless it is the first character entered in text mode.

The following example inserts three lines of text into the buffer by entering text mode. Note that Edit-11 signifies its return to command mode by printing an *.

```
*I <CR>

 BLOOP:      MOV     #17ØØØ,(R3)+      <CR>
             BIT     (R4)+,MASK        <CR>
             BNE     BLOOP             <CR>

 <LF>
*
```

The commands which require text objects are Insert, Get, wHole, Edit wHole, Position, Edit Position, Change and eXchange. (Capitalized letters represent the actual commands typed.)

CHAPTER 3

COMMAND SYNTAX

3.1 THE CHARACTER LOCATION POINTER (DOT)

Almost all Edit-11 commands function with respect to a movable reference point, Dot. This character pointer is normally located between the most recent character operated upon and the next character; and, at any given time, can be thought of as "where Edit-11 is" in your text. As will be seen shortly, there are commands which move Dot anywhere in the text, thereby redefining the "current location" and allowing greater facility in the use of the other commands.

3.2 MARK

In addition to Dot, a secondary character pointer known as Mark also exists in Edit-11. This less agile pointer is used with great effect to Mark or "remember" a location by moving to Dot and conditionally remaining there while Dot moves on to some other place in the text. Thus it is possible to think of Dot as "here" and Mark as "there". Positioning of Mark, which is referenced by means of the argument @, is discussed below in several commands.

3.3 CHARACTER-ORIENTED COMMAND PROPERTIES

Many Edit-11 commands are character-oriented; that is, the argument to the command specifies the number of characters in the Page Buffer the command is to act on. Hence, the number of characters specified by the argument n is the same in the forward (n) and backward (-n) direction. Carriage Return and line feed characters embedded between text lines are counted in character-oriented commands, and are indistinguishable from other characters.

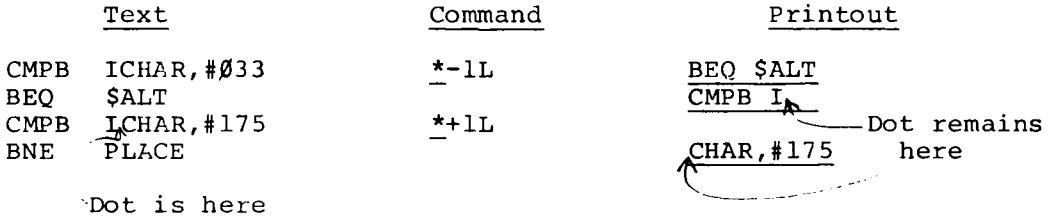
3.4 LINE-ORIENTED COMMAND PROPERTIES

Edit-11 recognizes a line as a unit by detecting a line terminator in the text. This means that ends of lines (line feed or form feed characters) are counted in line-oriented commands. This is important to know, particularly if Dot, which is a character location pointer, is not pointing at the first character of a line.

In such a case, an argument n will not affect the same number of lines (forward) as its negative (backward). For example, the argument

-1 applies to the character string beginning with the first character following the second previous end-of-line character and ending at Dot. Argument +1 applies to the character string beginning at Dot and ending at the first end-of-line character. If Dot is located, say, in the center of a line, notice that this would affect 1-1/2 lines back or 1/2 line forward respectively.

Example of List Commands -1L and +1L:



3.5 THE PAGE UNIT OF INPUT

Input files to Edit-11 are divided into smaller, more manageable segments called "pages". A page is terminated, and therefore defined, by a form feed (CTRL/FORM on keyboard) in the source text whenever a page division is desired. Although the unit of output is the line, the unit of input to Edit-11 is the page, and in order to make an ASCII file more usable, the user divides it into small segments by inserting form feeds in desired places before output occurs. Since more than one page of text can be in the buffer at the same time, it should be noted that the entire contents of the Page Buffer are available for editing.

3.6 ARGUMENTS

Some Edit-11 commands require an argument to specify the particular portion of text to be affected by the command or how many times to perform the command. In other commands this specification is implicit and arguments are not allowed.

The Edit-11 command arguments are described as follows:

1. n stands for any number from 1 to 32767 decimal and may, except where noted, be preceded by a + or -. If no sign precedes n, it is assumed to be a positive number.

   Where an argument is acceptable, its absence implies an argument of 1 (or -1 if a - is present). The role of n varies according to the command it is associated with.

2. Ø refers to the beginning of the current line.

3. @ refers to a Marked (designated) character location (see Section 3.2).

4. / refers to the end of text in the Page Buffer.

The roles of all arguments will be explained further with the corresponding commands which qualify them.

3.7 COMMAND STRINGS

A command string to Edit-11 consists of one or more commands typed on the same line. Spaces are not allowed between a command and its associated argument, although spaces may be inserted between commands themselves in the command string. The command string, including embedded text objects, must be less than 72 characters and is terminated by typing the RETURN key.

NOTE

Caution must be exercised when using spaces in text commands. If a space separates a text command from its object, the space is considered the delimiter. Hence:

1. I OBJECT B /L is legal.

2. I #OBJECT#B /L will use #OBJECT#B as the text object.

3. I !OBJECT!B/L is illegal because there is no second delimiter.

The following are all legal command strings:

```
*B /L B G#OBJECT# <CR>
*B/L BG#OBJECT#I#TEXT# B/L  <CR>
*BSKB3L  <CR>
*-3J -4C#TEXT# ØA L  <CR>
```

When the user types a command string to Edit-11, nothing happens until he types the RETURN key. Upon receipt of the return character, Edit-11 starts at the beginning of the command string and processes the commands one at a time. Should an error be encountered in the middle of the string, those commands before the error would be successfully completed, and those following the command in error would not be executed.

3.8 ERROR MESSAGES

Whenever Edit-11 receives a syntactically incorrect command, or cannot execute a command properly, it returns an error message of the form:

Xnnn	where X is either an S (System error) or W (Warning) and nnn is a numerical error code. (See Appendix A.)

If the error occurs after the first command in a command string, Edit-11 will follow the error code with a second line. The second line will be a copy of the command string being executed, with a ? following the character where Edit-11 found the error. The commands printed completely were executed and the commands not printed were not executed. Edit-11 then prints an * and awaits another command string.

Example:

```
*B /L G#TEXT# 6EF I#MORE#   <CR>
W314 ØØØØØØ
B /L G#TEXT# 6EF?
*
```

In most cases, Edit-11 performs no further action on receipt of an illegal command. In effect, the command string is truncated at the error point.

CHAPTER 4

INPUT/OUTPUT

When Edit-11 is initialized, it asks for a command string for the Command String Interpreter which serves to specify the input and output devices and files it will use. In this command string, the user must specify a primary output file, and may specify a primary input file and secondary input and output files if desired. (See section 6.1.)

4.1 PRIMARY INPUT AND OUTPUT

If two input files are specified, the first one specified is the primary input file. This file is the ASCII file you want to edit, and can only be read. The first output file specified is the primary output file and it is into this file that corrected text will be written. Essentially, Edit-11 will transform the primary input file into the primary output file under user direction. If the primary input and output files have the same name and are on the same device, Edit-11 creates the file EDITOR.TMP for output. When this file is closed (by the EX or EF command), the input file is renamed with .BAK as its extension. The output file, EDITOR.TMP is then renamed to the desired name. This backup facility is automatic, but it can be suppressed with the /B switch. (See Section 6.1.)

NOTE

Do not create a file of your own with the name EDITOR.TMP. It will be deleted the next time Edit-11 needs the name.

4.2 SECONDARY INPUT AND OUTPUT

If a second input or output file (or both) is specified, Edit-11 will allow text to be read or written from these files also. These files are utility files, designed to ease the process of editing. They are controlled by commands distinct from primary input and output commands. All secondary input/output commands begin with the letter E (ER, EW, EH, EP), while all primary input/output commands are one-letter commands.

CHAPTER 5

COMMANDS

5.1 INPUT AND OUTPUT COMMANDS

Five commands are available for reading in a page of text: The Read and Edit Read commands (section 5.1.1) are specialized input commands; the Next command (section 5.1.3) reads in a page after writing out the previous page; and the wHole and Edit wHole commands (sections 5.2.2.2 and 5.2.2.3) read in and write out text as part of a search for a specified character string.

Output commands either list text on the teleprinter or write text into an output file. The List command causes specified lines of text to be output to the teleprinter so that they may be examined. Write commands (Next, wHole, and Edit wHole also perform input), provide for the output of specified pages, lines, form feeds (for changing the amount of data that constitutes a given page), and leader/trailer for paper tape. Note that the process of outputting text does not cause Dot to move.

5.1.1 Read and Edit Read

Two ways of getting text into storage so that it can be edited are by means of the Read (R) and Edit Read (ER) commands. The command R causes a page of text to be read from the primary input file, while the command ER reads a page from the secondary input file. The read text is appended to the contents (if any) of the Page Buffer.

Text will be read in until either:

1. A form feed character is encountered;
2. The Page Buffer is 128 characters from being filled, or a line feed is encountered after the buffer has become 5ØØ characters from being full;
3. An end of data is detected on the input device.

Following executing of an R command, Dot and Mark will be located at the beginning of the Page Buffer.

An 8K system can accommodate about 4ØØØ characters of text. All

additional core memory is available for text storage, i.e., about 8ØØØ characters of text for each additional core memory bank.

NOTE

An attempt to overflow the storage area will cause the command (in this case, R) to stop executing. A W3Ø3 error message will be printed. No data will be lost.

5.1.2 List, Write and Edit Write

Output commands List, Write, and Edit Write can be described together, as they differ only in the device and file addressed. List (L) outputs to the teleprinter, Write (W) outputs to the primary output file, and Edit Write (EW) outputs to the secondary output file.

nL nW nEW	Lists Writes on primary output Writes on secondary output	the character string beginning at Dot and ending with the n^{th} end of line.
-nL -nW -nEW	Lists Writes on primary output Writes on secondary output	the character string beginning with the first character following the (n+1)th previous end of line and terminating at Dot.
ØL ØW ØEW	Lists Writes on primary output Writes on secondary output	the character string beginning with the first character of the current line and ending at Dot.
@L @W @EW	Lists Writes on primary output Writes on secondary output	the character string between Dot and the Marked location.
/L /W /EW	Lists Writes on primary output Writes on secondary output	the character string beginning at Dot and ending with the last character in the Page Buffer.

In addition to the above commands, there is a special list command that accepts no arguments.

V (Verify) lists the entire line containing Dot.

5.1.3 Next

Typing nN writes, onto the primary output file, the entire contents

of the Page Buffer (followed by trailer if paper tape is the output medium and a form feed is the last character in the buffer), deletes the contents of the buffer, and reads the Next page from the primary input file into the buffer. It performs this sequence n times. If there are fewer than the n pages specified, the command will be executed for the number of pages actually available, and error code W311 will be printed out. Following execution, Dot and Mark will be located at the beginning of the Page Buffer.

5.1.4 Form Feed and Trailer

F	Writes a Form feed character and four inches of null characters into the primary output file if the primary output is paper tape.
nT	Writes four inches of Trailer (null characters) n times on the primary output device if the primary output is paper tape.

5.2 EDITING COMMANDS

5.2.1 Commands to Move Dot and Mark

5.2.1.1 Beginning

B	moves Dot to the beginning of the Page Buffer

5.2.1.2 Jump and Advance

nJ	moves Dot forward past n characters.
nA	moves Dot forward past n ends-of-lines to the beginning of the succeeding line.
-nJ	moves Dot backward past n characters.
-nA	moves Dot backward to the first character following the (n+1)th previous end-of-line.
ØJ or ØA	moves Dot to the beginning of the current line
@J or @A	moves Dot to the Marked location.
/J or /A	moves Dot to the end of the Page Buffer

Notice that while n moves Dot n characters in the Jump command, its role becomes that of a line counter in the Advance command, However, because Ø, @, and / are absolute, their use with these commands overrides line/character distinctions. That is, Jump and Advance perform identical functions if both have either Ø, @ or / for an argument.

5.2.1.3 Mark

The M command Marks ("remembers") the current position of Dot for later reference in a command using the argument @. Note that only one position at a time can be in a marked state. Mark is also affected by the execution of the following commands:

C D H I K N R X EH ER

5.2.2 Search Commands

5.2.2.1 Get

The basic search command nG starts at Dot and Gets the n^{th} occurrence of the specified text object in the Page Buffer. If no argument is present, it is assumed to be 1. If the object is to follow the G command in the command string, it must be properly set off by delimiters. If G is the last character typed in the command string, Edit-11 will enter text mode and accept a search object of up to 72 characters.

This command sets Dot to the position immediately following the found character string.

If the search is unsuccessful, Dot will be at the end of the Page Buffer and W3Ø7 will be printed on the teleprinter before Edit-11 prints an *.

Examples:

1.

Text	Command	Effect
MOV @RMAX,@R5	2G <CR>	BEQ CK
ADD #6,(R5)+	CK <LF>	
CLR $CK3		
TST R2		
BEQ CKCR		
Dot was here		Dot is now here

2.

CMPB	ICHAR,#RUBOUT	G<CR>	BR
BEQ	SITE	TE<CR>	
BR	PUT	BR<LF>	
Dot			Dot

5.2.2.2 wHole

A second search command, nH, starts at Dot and looks through the wHole text file for the n^{th} occurrence of the character string you have specified as text object. It combines a Get and a Next such that if the search is not successful in the Page Buffer, the contents of the buffer

are written in the primary output file, the buffer contents are deleted, and a new page is read in from the primary input, where the search is continued. This will proceed until the search object is found or until the complete source text has been searched. In either case, Mark will be at the beginning of the Page Buffer.

If the search object is found, Dot will be located immediately following it. As in the Get command, if the search is not successful, Dot will be at the end of the buffer and a W3Ø7 will appear on the teleprinter. Upon completion of the command, Edit-11 will be in command mode. Note that an H command specifying a nonexistent search object can be used to close out an edit, i.e., copy all remaining text from the primary input file to the primary output file.

5.2.2.3 Edit wHole

A wHole search can be performed through the secondary input file with the EH command. The EH search is identical to the wHole command, except that the file searched is the secondary input, while the file written into is the primary output. Note that an EH search for a nonexistent object is a method for reading the entire secondary input file into the center or onto the end of the primary output file.

5.2.2.4 Position

A fourth search command, nP, is used to Position Edit-11 in the primary input file so that a Read command can read the desired page. Position combines the Get, Delete and Read commands so that if the search object is not found the Page Buffer is cleared and a new page is read from the primary input and searched. This process continues until the search object is found or the end of the file is reached. The Position search is a wHole search in which there is destruction of unsuccessfully searched pages.

The Position command works as follows:

1. The current buffer contents, if any, are output to the primary output device and the buffer is cleared.
2. A page is read into the buffer and searched.
3. If the search is unsuccessful, the Page Buffer is cleared and step 2 is repeated.
4. If the n^{th} occurrence is found, Edit-11 returns with the page in which the n^{th} occurrence resides in the buffer, with Dot following the search object and Mark at the beginning.

Note that before the search begins, the current Page Buffer is saved by Edit-11 automatically in the primary output file. If this buffer is not desired, it must be deleted by the user prior to searching.

Note also that the Position command is most useful as a means of placing Edit-11 in the input file. If the editing session wishes to create a new file out of the second half of the input file, a Position search will save time.

5.2.2.5 Edit Position

The nEP command is identical to the Position command, except that the input file read from and searched is the secondary input file. Note that any text in the buffer is output to the primary output file before the search begins.

5.2.3 Commands to Modify the Text

5.2.3.1 Insert

The Insert command (I) allows text to be inserted at Dot. After I is typed, Edit-11 needs the text object to be inserted. If text mode is used, up to 80 characters per line are acceptable, and execution of the command occurs when the LINE FEED key (which does not insert a line feed character unless it is the first character typed in text mode) is typed terminating text mode. At this point, Dot is located in the position immediately following the last inserted text character. If the Marked location was anywhere after the text to be inserted, Dot becomes the new Marked location.

As with the Read command, an attempt to overflow the Page Buffer will cause a W3Ø3 to be printed out followed by an * on the next line indicating that a command may be typed. All or part of the last typed line may be lost. All previously typed lines will be inserted.

Examples:

	Text	Command	Effect
1.	MOV #8.,EKOT Dot	I <CR> CN <LF>	MOV #8.,EKOCNT Dot
2.	Inserting a carriage return (and automatic line feed):		
	CLR R1CLR R2 Dot	I <CR> <CR> <LF>	CLR R1 CLR R2 Dot
3.	Inserting a single line feed:		
	LOOK WHAT HAPPENS HERE Dot	I<CR> <LF> <LF>	LOOK WHAT HAPPENS HERE Dot

5.2.3.2 Delete and Kill

The commands in this category are closely related to each other; they both erase specified text from the Page Buffer. The Delete command differs from the Kill command only in that the former accepts an argument, n, that counts characters to be removed, while the latter accepts an argument, n, that counts lines to be removed. Ø, @ and / are also allowed as arguments (see Section 3.6). After execution of either of these commands, Dot becomes the Marked location.

nD	Deletes the following n characters	nK	Kills the character string beginning at Dot and ending at the n^{th} end-of-line
-nD	Deletes the previous n characters	-nK	Kills the character string beginning with the first character following the (n+1)th previous end-of-line and ending at Dot.

ØD or ØK — Removes the current line up to Dot

@D or @K — Removes the character string bounded by Dot and Mark

/D or /K — Removes the character string beginning at Dot and ending with the last character in the Page Buffer.

Examples:

	Text	Command	Effect
1.	;CHECK THE MOZXDE Dot	-2D	;CHECK THE MODE Dot
2.	;IS IT A TAB OR ;IS IT A CR Dot	2K	;IS IT A TAB Dot

5.2.3.3 Change and eXchange

The Change (C) and eXchange (X) commands can be thought of as two-phase commands combining, respectively, an Insert followed by a Delete, and an Insert followed by a Kill. After the Change or eXchange command is typed, a text object is required to be inserted. If ±n is used as the argument, it is then interpreted as in the Delete (character-oriented) or Kill (line-oriented) commands, and accordingly removes the indicated text. Ø, @, and / are also allowed in arguments.

commands combining, respectively, an Insert followed by a Delete, and an Insert followed by a Kill. After the Change or eXchange command is typed, a text object is required to be inserted. If ±n is used as the argument, it is then interpreted as in the Delete (character-oriented) or Kill (line-oriented) commands, and accordingly removes the indicated text. Ø, @, and / are also allowed as arguments.

nC xxxx xxxx	Changes the following n characters	nX xxxx xxxx	eXchanges the character string beginning at Dot and ending at the n^{th} end-of-line
-nC xxx	Changes the previous n characters	-nX xxx	eXchanges the character string beginning with the first character following the (n+1)th previous end-of-line and ending at Dot

ØC xxxx xxxx	or	ØX xxxx xxxx	Replaces the current line up to Dot
@C xxx xxx	or	@X xxx xxx	Replaces the character string bounded by Dot and the Marked location
/C xxx	or	/X xxx	Replaces the character string beginning at Dot and ending with the last character in the Page Buffer.

Again, the use of absolute arguments Ø, @, and / overrides the line/character distinctions that n and -n produce in these commands.

If the Insert portion of these commands is terminated because of attempting to overflow the Page Buffer, data from the latest line may have been lost, and text removal will not occur. Such buffer overflow might be avoided by separately executing a Delete or Kill followed by an Insert, rather than a Change or eXchange, which does an Insert followed by a Delete or Kill.

Examples:

Text	Command	Effect
;A LINE FEED IS HERE ;THIS ;IS ON ;FOUR ;LINES	-9C<CR> TAB<LF> 2X<CR> PAPER<LF>	;A TAB IS HERE ;THIS ;IS ON ;PAPER

Dot

Dot

5.2.4 Utility Commands

The following commands provide easier paths for more general editing functions.

5.2.4.1 Save

The nS command copies the n lines beginning at Dot into an external buffer called the Save Buffer. Dot does not change, nor is the saved text deleted. Any previous contents of the Save Buffer are destroyed. If there is not enough room in storage to copy the text specified, a W303 error message will be printed and none of the text will be saved. Note that Save is a line-oriented command, and operates only in the forward direction; that is, negative arguments are not allowed.

5.2.4.2 Unsave

The U command inserts the entire contents of the Save Buffer at Dot, and Dot moves to follow the last character Unsaved. The Save Buffer itself is retained, and the same Save Buffer can be Unsaved as often as desired. If the action of unsaving would result in overflow of the Page Buffer, a W31Ø error message is printed and the Unsave does not occur.

Note that Save and Unsave provide convenient tools for moving blocks of text or inserting the same block of text in several places.

5.2.4.3 Execute Macro

The nEM command is a means of performing the same Edit-11 command string n times. When the EM command is received, the contents of the Save Buffer up to the first <CR> character are interpreted as a command string and executed n times. The macro is subject to the same rules as any typed command string, and in addition, a macro string may not contain another macro call.

Thus, to execute a macro, it is required to insert the macro in the Page Buffer, save it, then execute.

Example:

The following sequence of commands will change the first 15 occurrences of .CSECT in the buffer to .ASECT.

```
* BI
  B G#.CSECT# -4J -C#A#<CR>
  <LF>
* BSK15EM
```

5.2.4.4 Edit Open

The EO command will close the secondary input file and re-open it at the beginning. Therefore, although Edit-11 is a one-pass editor, that is, capable of making only one trip through the primary input file per job, it is possible to make many passes through the subsidiary input file with the EO command. EO has no effect on the text.

5.2.4.5 End File

The EF command closes the primary output file to any further output, and renames it if a backup file is to be created. It also closes the primary input file to any further read operations. The secondary output and secondary input files remain open for further editing. Note that this command is useful to create a truncated output file from a large input file.

5.2.4.6 EXit

The EX command is the most common way to terminate an editing session. Exit copies the remainder of the primary input file into the primary output file, closes all files, and begins another edit.

If, however, an EF command was executed during the editing session, EX will not perform any input/output operations - it merely closes files. The last command of every editing session should be the EX command. Whether it involves any I/O or not depends on whether the primary I/O files are open or closed.

CHAPTER 6

OPERATING INSTRUCTIONS

6.1 STARTING

To start Edit-11, load the PDP-11 Disk Operating System (if it is not running already) and log in. Then type:

```
$RU(N) EDIT11
```

Upon receipt of this command, the DOS will load Edit-11 and begin its execution. The first thing Edit-11 needs is input/output specifications for the editing session. After being loaded and started, Edit-11 responds with #, indicating a request for a command string. The user responds with a command string of the form:

```
#dev:file1.ext[uic],dev:file2.ext[uic]<dev:file3.ext[uic]/B,dev:file4.ext[uic]
```

In the above specification, FILE1 is the primary output and FILE3 is the primary input, while FILE2 and FILE4 are the secondary output and input respectively. Note the following:

1. A minimum of one output must be specified, and a maximum of 2 input and 2 output may be specified. Hence, file2, file3, and file4 above are optional.

2. dev: is the device name and is optional; disk is assumed if none is specified. The assumed device changes, however, as soon as a new device is typed. See Disk Operating System Monitor Programmer's Handbook.

3. Filen.ext is the file name of the appropriate input or output file.

4. [uic] is the user code for the owner of the file, and need not be specified if the file in question is your file. Note that [(SHIFT/K) and] (SHIFT/M) are part of the uic specification and must be typed if a uic is specified.

5. /B is the backup switch. If /B is specified, no backup file will be created. Otherwise, the backup facility will be used (see section 4.1). Note that if the /B switch is used, it must follow the primary input file specification.

6. If file1 and file3 are the same file on one device, file3 will be renamed file3.BAK unless the /B switch is used. file2 and file4 cannot be the same file.

When Edit-11 receives a syntactically correct specification, it performs internal initialization, then types an *, indicating its readiness to receive the first command.

6.2 RESTARTING

To restart Edit-11, use the Monitor REstart command after issuing a CTRL/C.

```
*CTRL/C
.RE
*
```

Note that the above method of restarting enables you to stop a command execution if desired.

6.3 FINISHING AN EDIT

To finish an editing session, use the EX command. This will restart Edit-11 for the next job unless the keyboard (KB:) has been specified as a primary input device. In such a case, the EX must be followed by typing CTRL/C, EN, the RETURN key, and the LINE FEED key.

NOTE

Any other way of leaving an editing session will result in the input and output files not being properly closed. A file not properly closed cannot be used by any system program until correctly closed by the (Unlock) option in PIP. Therefore, if an edit is terminated without an EF or EX command (the plug for the computer was pulled out, the Monitor crashed, or you executed another Monitor command after a CTRL/C), the input and output files must be restored by PIP before attempting to edit again.

6.4 ERROR RECOVERY

In the course of editing a page of the program, it may become necessary to correct mistakes in the commands themselves. There are two special commands which do this:

1. Typing the RUBOUT key removes the preceding typed character, if it is on the current line.

2. The CTRL/U combination (holding down the CTRL key and typing U) removes all the characters in the current line.

If, in the course of a command string execution, a serious error is discovered, the restart capability can be used to terminate the command. (See section 6.2.)

6.5 PROCEDURE WITH LOW-SPEED PUNCH

If the low-speed punch is one of the output devices, Edit-11 pauses before executing any command to write on the punch. The punch must be turned on at this time, after which typing the LINE FEED key on the keyboard initiates the output (the key typed does not echo on the tape). Following completion of the operation, Edit-11 pauses again to allow you to turn the punch off. When the punch has been turned off, type the LINE FEED key and Edit-11 will return to command mode.

CHAPTER 7

IMPLEMENTATION NOTES

7.1 MACRO USAGE

Use of the EM macro is most efficient if the text involved is small enough to fit within the macro itself. Large amounts of text can be inserted by a macro, however, if use is made of the fact that the macro itself is only the first line of the Save Buffer.

For example, suppose it is desirable to insert

```
JSR     R5,RSAVE
JSR     R5,RELOAD
```

after every occurrence of EMT 4Ø in your program. The text to be inserted is too long to enter in your command mode, and there is no way to enter text mode from a macro. Use can be made, however, of the Unsave command.

To accomplish the above, the commands would look like:

```
*BI
G;EMT 4Ø; A U -3A K
     JSR    R5,RSAVE
     JSR    R5,RELOAD
<LF>
*B3S3K 5ØØØEM
```

The first command inserts the 3 lines we need to save into the Page Buffer. Note that the first of the 3 lines is the macro, while the last two are the common code. The macro itself contains the Get command to look for the EMT 4Ø, followed by an Advance command to advance Dot to the next line. The next command is the Unsave command, which will Unsave the three lines in the Save Buffer. Since the macro is by definition the first line of the Save Buffer, it is Unsaved along with the other two and has to be deleted. The -3A and Kill commands accomplish this. The second command string saves the macro and common text, deletes them from the Page Buffer, and executes the macro 5ØØØ times. Five thousand is an arbitrarily large number which assures that we get all the occurrences of EMT 4Ø in the buffer. When the search fails, the macro will be halted and an error returned as Edit-11 prepares to accept another user command string.

7.2 DELIMITER USAGE

When entering text in command mode, any ASCII character is acceptable as a delimiter. Some, however, are better than others, and here are some suggestions when using delimiters.

1. Use the same delimiter all the time (except when not possible because it appears in the text itself). If you pick an uncommonly used character that is easy to type, such as Q or ;, typing the delimiter will become second nature.

2. Avoid delimiters that are valid arguments or commands. Use of /, @ and valid commands as delimiters is an error-prone practice. Avoid spaces most of all, as forgetting that there is a space in the text string will result in the execution of undesired commands, and this is a common error.

   Example:

   ```
   *G SAVE /DUMMY/ I$TEXT$
   ```

   Although the user wanted to get "SAVE /DUMMY/", he will actually Get "SAVE", Delete to the end of the buffer, Unsave, Mark, Mark again, then receive an error message for the illegal Y command.

7.3 SUBSIDIARY I/O

Subsidiary I/O can serve several purposes. Here are some suggestions on how it might be used.

1. If your secondary output file is a line printer or scope, you can look at the entire Page Buffer very quickly. After typing in a long Insert, look at it via the B/EW command and check for mistakes.

2. Use of two output files is ideal for dividing a single input file into two smaller files.

3. If you want to move a very long section of text, or you want to save text for insertion later but also want the Save Buffer free, use the HSR and HSP as your subsidiary I/O. Punch the desired text, then read it in wherever desired.

4. Two input files are ideal for concatenation also; wHole search for a nonexistent object through the primary input, then do the same through secondary input.

Remember that to close subsidiary files, the EX command must be used.

7.4 SAVE AND UNSAVE

Edit-11 saves text during the Save command by setting aside a buffer large enough to accommodate the saved text. This decreases the total core available to the Page Buffer, resulting in the possibility that there might not be enough free core left to Unsave the text. Edit-11 guards against this situation by allowing you to save only text that is short enough to guarantee at least enough room to insert it again. That is, you are guaranteed there will be enough room to Unsave at least once following every Save command that completes successfully. Of course, if you read or Insert a large amount of text between Save and Unsave, you might decrease the space available too much and not be able to Unsave. In this case, part of the buffer will have to be written into the output file to make room. You will be safe if you do all your Unsaves immediately following the corresponding Save commands.

7.5 CREATING A NEW FILE

If the purpose of an editing session is to create a file for which there is no input file, a primary input device need not be specified. Inserts are used to create the file and EX to close it. However, if the keyboard is specified as a primary input device, the file should be closed by the following sequence:

1. Write the file
2. Issue the EF command
3. If the editing session is to be terminated, type the EX command

If you should accidentally issue a read command such as R, N, EX, you can terminate the read with a form feed character. If you type EX without a previous EF and the keyboard is your input device, you must type CTRL/C, wait for a . to be printed out, and then type EN, the RETURN key, and the LINE FEED key in order to proceed successfully.

7.6 USING EXIT

Always end the editing session with the EX command. EX will guarantee that all files are closed correctly. Recall that if you do not want EX to perform any input file copying or buffer writing, precede it with the EF command.

APPENDIX A

ERROR DIAGNOSTICS

Error Code	Meaning	Cause
S202	Device Full	Output device does not have sufficient room to continue.
S203	Switch Error	Too many switches or illegal switch. A switch appeared which was not equal to /B or which followed a file other than primary input; or more than one switch appeared.
S204	Too Many Output Files	More than two output files were specified or a switch appeared after secondary output.
S205	Too Many Input Files	More than two input files were specified or a switch followed the secondary input.
S227	Illegal File Specification	Argument indicates specific violation. 1 - No primary output specified 2 - Secondary input equals secondary output 3 - Secondary input equals primary output 4 - Primary input equals secondary output 5 - Primary input equals secondary input 6 - Primary output equals secondary output

Command Syntax Errors - Command syntax errors are reported by printing the command up to and including the character at which the scan terminated, followed by a question mark and vertical tab. This does not mean the last character typed is the cause of the syntax error.

W303	Buffer Overflow	Command Input Buffer, Text Input Buffer, Save Buffer or Page Buffer overflow.
W304	Macro Overflow	Macro as stored in Save Buffer is too long to execute.
W305	Recursive Macro	Macro contains an EM command.
W306	Empty Save Buffer	An EM or U command was issued with nothing in the Save Buffer.
W307	Search Failure	n^{th} occurrence of search object was not found in available text.

Error Code	Meaning	Cause
W310	No Room to Unsave	Not enough available room to unsave required text.
W311	End of Data	End of input medium or end of input file reached during read. Last page read was last in file.
W312	Illegal Line Feed	A line feed character was encountered in command string.
W313	Illegal Negative Argument	The command specified does not accept negative arguments.
W314	No Arguments Allowed	The command specified does not recognize any arguments.
W315	Illegal Argument	Command does not accept given argument.
W316	Illegal Text String	Usually caused by missing second delimiter.
W317	Illegal Command	Edit-11 cannot execute command as requested. Usually caused by secondary I/O commands when no secondary I/O was specified at initialization time.
W320	Page Buffer Almost Full	Page Buffer within 128 characters of being full.
W321	File Closed	Attempt to Read or Write primary files after EF.

APPENDIX B

COMMANDS

In the following table, # represents any legal text delimiter. <CR> represents a return character, and <LF> represents a line feed character.

Command	Format	Result
Read	R	Read from primary input file until form feed encountered.
Edit Read	ER	Read from secondary input file until form feed encountered.
Write	nW	Write n lines into primary output file.
Edit Write	nEW	Write n lines into secondary output file.
Form feed	F	Write form feed into primary output file.
Trailer	nT	Write 4Ø null characters as trailer on the primary output device if device is paper tape.
Next	nN	Write the contents of the Page Buffer onto the primary output file, kill the buffer, and read a page of text from the primary input file. Repeat n times. Equivalent to B/W /D R.
Beginning	B	Move Dot to the beginning of the Page Buffer.
Advance	nA	Advance Dot n lines. Leaves Dot at beginning of line.
Jump	nJ	Move Dot over n characters.
Delete	nD	Delete n characters from text.
Kill	nK	Kill n lines of text.

Command	Format	Result
Mark	M	Mark the current location of Dot.
Save	nS	Save the next n lines in the Save Buffer.
Unsave	U	Copy the contents of the Save Buffer into Page Buffer at Dot.
List	nL	List n lines on teleprinter
Verify	V	Verify the present line via teleprinter.
Get	nG#XXXXX# or nG<CR> XXXX<CR> <LF>	Search for the n^{th} occurrence of XXXXX. Return with Dot following XXXXX.
wHole	nH#XXXXX# or nH<CR> XXXXX<CR> <LF>	Search for the n^{th} occurrence of XXXXX. If found, return with Dot following XXXXX. If not found, execute an N command and continue search.
Edit wHole	nEH#XXXXX# or nEH<CR> XXXX <LF>	Perform a wHole search for the n^{th} occurrence of XXXXX, using the secondary input and primary output files.
Position	nP#XXXXX# or nP<CR> XXXX <LF>	Perform a Next command, then search for the n^{th} occurrence of XXXXX. If found, return with Dot following XXXX. If not found, clear the buffer, read another page, and continue search.
Edit Position	nEP#XXXXX# or nEP<CR> XXXX <LF>	Perform a Position search using secondary input rather than primary input file.
Insert	I#XXXXX# or I<CR> XXXXX<CR> <LF>	Insert the text XXXXX at Dot. Move Dot to follow XXXXX.
Change	nC#XXXXX# <LF> or nC<CR> XXXXX<CR> <LF>	Change n characters to XXXXX. Equivalent to Insert followed by n Delete
eXchange	nX#XXXXX# or nX<CR> XXXXX<CR> <LF>	eXchange n lines for XXXXX. Equivalent to Insert followed by n Kill.

Command	Format	Result
Execute Macro	nEM	Execute the first line of the Save Buffer as a command string n times.
Exit	EX	Perform consecutive Next commands until EOM or EOF reached. Close all files, and return to Monitor.
Edit Open	EO	Move to the beginning of the secondary input file. Must follow a W311 error message before an ER can be executed.
End File	EF	Close the primary output file to any further output and close the primary input file.

Arguments, 1-1, 3-1, 3-2, 5-3, 5-7, 5-8

Backup facility, 4-1
Buffer capacity, 5-2, 5-6, 5-8, 5-9

Closing files, 6-2
Command mode, 2-1, 2-2, 6-3
Command string, 1-1, 2-1, 3-3, 3-4, 4-1, 5-9, 6-1, 7-1

Delimiter, text, 2-1, 3-3, 5-4, 7-2
Dot, 3-1, 5-1, 5-3 through 5-7, 5-9

Errors, 3-3, 3-4, 5-2 through 5-6, 6-2, 7-1, 7-2, A-1

Files,
  closing, 6-2
  primary input, 4-1, 5-1, 5-5, 5-9, 5-10, 6-1, 6-2, 7-2, 7-3
  primary output, 4-1, 5-2, 5-5, 5-6, 5-10, 6-1
  secondary input, 4-1, 5-1, 5-5, 5-6, 5-9, 5-10, 6-1, 7-2
  secondary output, 4-1, 5-2, 5-10, 6-1, 7-2

High-speed reader, 7-2

Input/Output commands, 5-1

Keyboard, 7-3

Line capacity for insert, 5-6
Line terminators, 3-1
Low-speed punch, 6-3

Macro, 5-9, 7-1
Mark, 3-1, 5-1, 5-3 through 5-7
Modes
  command, 2-1, 2-2, 6-3
  text, 2-1, 5-4, 5-6

Operating procedures, 6-1
Overflow, 5-2, 5-6, 5-8, 5-9

Page buffer, 1-1, 3-1, 3-2, 3-3, 5-1, 5-3 through 5-9, 7-1, 7-2, 7-3
Pages, 3-2, 5-1, 5-3
Primary input file, see Files
Primary output file, see Files

Save buffer, 5-9, 7-1, 7-2
Searches, 5-4, 5-5, 7-1
Secondary input file, see Files
Secondary output file, see Files

Terminators, line, 3-1
Text delimiter, 2-1, 3-3, 5-4, 7-2
Text mode, 2-1, 5-4, 5-6
Text object, 1-1, 2-1, 2-2, 3-3 5-4, 5-6, 5-7, 5-8

DEC-11-OODA-D

PDP-11

ODT-11R DEBUGGING PROGRAM

PROGRAMMER'S MANUAL

for the

Disk Operating System

SOFTWARE SUPPORT CATEGORY

The software described in this document
is supported by DEC under Category I,
as defined on page iv of this document.

For additional copies, order No. DEC-11-OODA-D from
Digital Equipment Corporation, Direct Mail, Bldg. 1-1,
Maynard, Mass. 01754

First Printing, May 1971
Second Printing, September 1971

Your attention is invited to the last two pages of this document. The "How to Obtain Software Information" page tells you how to keep up-to-date with DEC's software. The "Reader's Comments" page, when filled in and mailed, is beneficial to both you and DEC; all comments received are acknowledged and are considered when documenting subsequent manuals.

Copyright (C) 1971 by Digital Equipment Corporation

This document is for information purposes and is subject to change without notice.

Associated PDP-11 Documents:

Disk Operating System Monitor,
Programmer's Handbook, DEC-11-MWDC-D

PAL-11R Assembler,
Programmer's Manual, DEC-11-ASDC-D

FORTRAN IV,
Programmer's Manual, DEC-11-KFDA-D

Edit-11 Text Editor,
Programmer's Manual, DEC-11-EEDA-D

ODT-11R Debugging Program,
Programmer's Manual, DEC-11-OODA-D

PIP-11, File Utility Package,
Programmer's Manual, DEC-11-PIDB-D

Link-11 Linker and Libr-11 Librarian,
Programmer's Manual, DEC-11-ZLDC-D

The following are trademarks of Digital Equipment Corporation:

DEC	PDP
FLIP CHIP	FOCAL
DIGITAL (logo)	COMPUTER LAB
OMNIBUS	UNIBUS

P R E F A C E

This manual describes the features and operation of the ODT-11R Debugging Program for the PDP-11 Disk Operating System (DOS). The manual assumes familiarity with the Disk Operating System Monitor (see PDP-11 Disk Operating System Monitor, Programmer's Handbook, DEC-11-MWDC-D).

In addition to the ODT-11R Debugging Program and the DOS Monitor, the Disk Operating System software includes:

PAL-11R Assembler
Edit-11 Text Editor
PIP-11, File Utility Package
Link-11 Linker
Libr-11 Librarian
FORTRAN IV

SOFTWARE SUPPORT CATEGORIES

Digital Equipment Corporation (DEC) makes available four categories of software. These categories reflect the types of support a customer may expect from DEC for a specified software product. DEC reserves the right to change the category of a software product at any time. The four categories are as follows:

CATEGORY I
Software Products Supported at no Charge

This classification includes current versions of monitors, programming languages, and support programs provided by DEC. DEC will provide installation (when applicable), advisory, and remedial support at no charge. These services are limited to original purchasers of DEC computer systems who have the requisite DEC equipment and software products.

At the option of DEC, a software product may be recategorized from Category I to Category II for a particular customer if the software product has been modified by the customer or a third party.

CATEGORY II
Software Products that Receive Support for a Fee

This category includes prior versions of Category I programs and all other programs available from DEC for which support is given. Programming assistance (additional support), as available, will be provided on these DEC programs and non-DEC programs when used in conjunction with these DEC programs and equipment supplied by DEC.

CATEGORY III
Pre-Release Software

DEC may elect to release certain software products to customers in order to facilitate final testing and/or customer familiarization. In this event, DEC will limit the use of such pre-release software to internal, non-competitive applications. Category III software is only supported by DEC where this support is consistent with evaluation of the software product. While DEC will be grateful for the reporting of any criticism and suggestions pertaining to a pre-release, there exists no commitment to respond to these reports.

CATEGORY IV
Non-Supported Software

This category includes all programs for which no support is given

C O N T E N T S

			Page
CHAPTER 1		INTRODUCTION	1-1
	1.1	Relocation	1-2
	1.2	Relocatable Expressions	1-2
	1.3	Commands	1-4
CHAPTER 2		COMMANDS AND FUNCTIONS	2-1
	2.1	Printout Formats	2-1
	2.2	Opening, Changing, and Closing Locations	2-2
	2.2.1	The Slash, /	2-2
	2.2.2	The Backslash, \	2-3
	2.2.3	The LINE FEED key, ↓	2-4
	2.2.4	The Up-Arrow, ↑	2-4
	2.2.5	The Back-Arrow, ←	2-4
	2.2.6	Opening the Addressed Location, @	2-5
	2.2.7	Relative Branch Offset, >	2-5
	2.2.8	Return to Previous Sequence, <	2-5
	2.2.9	Accessing General Registers 0-7	2-6
	2.2.10	Accessing Internal Registers	2-6
	2.2.11	Entering Radix50 Mode, X	2-7
	2.3	Breakpoints	2-8
	2.4	Running the Program, r;G and r;P	2-10
	2.5	Single-Instruction Mode	2-11
	2.6	Searches	2-12
	2.6.1	Word Search, r;W	2-12
	2.6.2	Effective Address Search, r;E	2-13
	2.7	The Constant Register, r;C	2-13
	2.8	Core Block Initialization, ;F and ;I	2-14
	2.9	Calculating Offsets, r;O	2-15
	2.10	Relocation Register Commands, r;nR, ;nR ;R	2-16
	2.11	The Relocation Calculators, nR and n!	2-17
	2.12	ODT's Priority Level, $P	2-18
	2.13	ASCII Input and Output, r;nA	2-19
	2.14	Return to Monitor, CTRL/C	2-19

C O N T E N T S

		Page
CHAPTER 3	ERROR DETECTION	3-1
CHAPTER 4	PROGRAMMING CONSIDERATIONS	4-1
4.1	Functional Organization	4-1
4.2	Breakpoints	4-1
4.3	Search	4-6
4.4	Teletype Interrupt	4-7
CHAPTER 5	OPERATING PROCEDURES	5-1
5.1	Loading Procedures	5-1
5.2	Starting and Restarting	5-1

CHAPTER 1

INTRODUCTION

ODT-11R (On-line Debugging Technique for the PDP-11) is a system program which aids in debugging assembled and linked object programs. From the keyboard you interact with ODT and the object program to:

- print the contents of any location for examination or alteration,
- run all or any portion of your object program using the breakpoint feature,
- search the object program for specific bit patterns,
- search the object program for words which reference a specific word,
- calculate offsets for relative addresses,
- fill a block of words or bytes with a designated value.

During a debugging session you should have at the teleprinter the assembly listing of the program to be debugged. Minor corrections to the program may be made on-line during the debugging session. The program may then be run under control of ODT to verify any change made. Major corrections, however, such as a missing subroutine should be noted on the assembly listing and incorporated in a subsequent updated program assembly.

1.1 RELOCATION

When the relocatable assembler produces a binary relocatable object module, the base address of the module is taken to be location 000000, and the addresses of all program locations as shown in the assembly listing are indicated relative to this base address. After the module is linked by the Linker, many values within the program, and all the addresses of locations in the program, will be incremented by a constant whose value is the actual absolute base of the module after it has been relocated. This constant is called the relocation bias for the module. Since a linked program may contain several relocated modules, each with its own relocation bias, and since, in the process of debugging, these biases will have to be subtracted from absolute addresses continually in order to relate relocated code to assembly listings, ODT-11R provides an automatic relocation facility.

The basis of the relocation facility lies in 8 relocation registers, numbered 0 through 7, which may be set to the values of the relocation biases you are interested in at a given time during debugging. Relocation biases should be obtained by consulting the memory map produced by the Linker. Once set, a relocation register is used by ODT-11R to relate relocatable code to relocated code. For more information on the exact nature of the relocation process, consult the manual on the Link-11 Linker (DEC-11-ZLDA-D).

1.2 RELOCATABLE EXPRESSIONS

The symbol n below stands for an integer in the range 0 to 7 inclusive.

The symbol k stands for an octal number up to six digits long, with a maximum value of 177777. If more than six digits are typed, ODT-11R takes the last six digits, truncated to the low-order 16 bits. k may be preceded by a minus sign, in which case its value will be the two's complement of the number typed. For example:

k (number typed)	Value
1	000001
-1	177777
400	000400
-177730	000050
1234567	034567

The symbol r is called a relocatable expression and is evaluated by ODT-11R as a 16-bit (6 octal digit) number. It may be typed in any one of three forms:

Form A	k	The value r is simply the value of k.
Form B	n,k	The value of r is the value of k plus the contents of relocation register n. If the n part of this expression is greater than 7, ODT takes only the last octal digit of n.
Form C	C or C,k or n,C or C,C	Whenever the letter C is typed, ODT-11R replaces C with the contents of a special register called the Constant Register. This value has the same role as the k or n that it replaces. The Constant Register is designated by the symbol $C and may be set to any value, as indicated below.

In the following examples, assume that relocation register 3 contains 003400 and that the constant register contains 000003.

r	Value of r
5	000005
-17	177761
3,0	003400
3,150	003550
3,-1	003377
C	000003
3,C	003403
C,0	003400
C,10	003410
C,C	003403

NOTE

For simplicity's sake, most examples in this section use Form A. All three forms of r are equally acceptable, however.

1.3 COMMANDS

ODT's commands are composed using the following characters and symbols. They are often used in combination with the address upon which the operation is to occur, and are offered here for familiarization prior to their thorough coverage which follows.

r/	open the word at location r.
/	reopen the last opened location.
r\(SHIFT/L)	open the byte at location r.
\	reopen the last opened byte.
nR	after a word has been opened, retype the contents of the word relative to relocation register n—i.e., subtract contents of relocation register n from the contents of the opened word and print the result. If n is omitted, ODT-11R selects the relocation register whose contents are closest but less than or equal to the contents of the opened location.
n!	after a word or byte has been opened, print the address of the opened location relative to relocation register n. If n is omitted, ODT-11R selects the relocation register whose contents are closest, but less than or equal to the address of the opened location.
↓ (LINE FEED key)	open next sequential location.
↑ or ^	open previous location[1]
RETURN	close open location and accept the next command
← or _	take contents of opened location, index by contents of PC, and open that location.[2]

[1] The circumflex, ^, appears on some keyboards and prints in place of the up-arrow.

[2] The underline, _, appears on some keyboards and prints in place of the back-arrow.

@	take contents of opened location as absolute address and open that location.
>	take contents of opened location n as relative branch instruction and open referenced location.
<	return to sequence prior to last @, >, or ↑ command and open succeeding location.
X	perform a Radix 50 unpack of the binary contents of the current opened word; then permit the storage of a new Radix 50 binary number in the same location.
r;O	calculate offset from currently open location to r.
$n/	open general register n (0-7)
$y/	open special register y, where y may be one of the following letters: S Status register (saved by ODT after a breakpoint) M Mask register B First word of the breakpoint table P Priority register C Constant register R First relocation register (register 0) F Format register
;F	fill memory words with the contents of the constant register.
;I	fill memory bytes with the contents of the low-order 8 bits of the constant register.
;	separate commands from command arguments (used with alphabetic commands below); separate a relocation register specifier from an addend.
;B	remove all Breakpoints
r;B	set Breakpoint at location r
r;nB	set Breakpoint n at location r
;nB	remove n^{th} Breakpoint
r;E	search for instructions that reference Effective address r
r;W	search for Words with bit patterns which match r

;nS	enable Single-instruction mode (n can have any value and is not significant); disable breakpoints
;S	disable Single-instruction mode; reenable breakpoints.
r;G	Go to location n and start program run
;P	Proceed with program execution from breakpoint; stop when next breakpoint is encountered or at end of program In Single-instruction mode only, Proceed to execute next instruction only
k;P	Proceed with program execution from Breakpoint; stop after encountering the Breakpoint k times In Single-instruction mode only, Proceed to execute next k instructions.
;R	set all relocation registers to -1 (highest address value).
;nR	set relocation register n to -1.
r;nR	set relocation register n to the value of r. If n is omitted, it is assumed to be 0
r;C	print the value of r and store it in the constant register
r;nA	print n bytes in their ASCII format, starting at location r; then allow n bytes to be typed in, starting at location r.
CTRL/C	prepare Monitor to accept a command from the keyboard.

CHAPTER 2

COMMANDS AND FUNCTIONS

When ODT is started as explained in Chapter 5, it will indicate its readiness to accept commands by printing an asterisk on the left margin of the teleprinter paper. In response to the asterisk you can issue most commands; for example, you can examine and, if desired, change a word, run the object program in its entirety or in segments, or even search core for certain words or references to certain words. The discussion below explains these features.

All commands to ODT are stated using the characters and symbols shown in Sections 1.2 and 1.3.

2.1 PRINTOUT FORMATS

Normally, when ODT prints addresses (as with the commands ↓, ↑, ←, @, <, and >) it attempts to print them in relative form (FORM B in Section 1.2). ODT looks for the relocation register whose value is closest but less than or equal to the address to be printed, and then represents the address relative to the contents of the relocation register. However, if no relocation register fits the requirement, the address is printed in absolute form. Since the relocation registers are initialized to -1 (the highest number) the addresses are initially printed in absolute form. If any relocation register subsequently has its contents changed, it may then, depending on the command, qualify for relative form.

For example, suppose relocation registers 1 and 2 contained 1000 and 1004 respectively, and all other relocation registers contained numbers much higher. Then the following sequence might occur:

```
*774/000000↓
000776/000000↓
1,000000 /000000↓
1,000002 /000000↓
2,000000 /000000
```

The format is controlled by the format register, $F. Normally this register contains 0, in which case ODT prints addresses relatively whenever possible. $F may be opened and changed to a non-zero value, however, in which case all addresses will be printed in absolute (see section 2.2.10).

2.2 OPENING, CHANGING AND CLOSING LOCATIONS

An open location is one whose contents ODT has printed for examination, and whose contents are available for change. A closed location is one whose contents are no longer available for change.

The contents of an open location may be changed by typing the new contents followed by a single character command which requires no argument (i.e.,↓, ↑, RETURN, ←, @, >, <). Any command typed to open a location when another location is already open, will first cause the currently open location to be closed.

2.2.1 The Slash /

One way to open a location is to type its address followed by a slash:

```
*1000/012746
```

Location 1000 is open for examination and is available for change. Note that in all examples ODT's printout is underlined; your typed input is not.

Should you not wish to change the contents of an open location, merely type the RETURN key and the location will be closed; ODT will print another asterisk and wait for another command. However, should you wish to change the word, simply type the new contents before giving a command to close the location:

```
*1000/012746 012345
*
```

In the example above, location 1000 now contains 012345 and is closed since the RETURN key was typed after entering the new contents, as indicated by ODT's second asterisk. Used alone, the slash

will reopen the last location opened:

```
*1000/012345 2340
*/002340
```

In the example above, the open location was closed by typing the RETURN key. ODT changed the contents of location 1000 to 002340 and then closed the location before printing the *. We then typed a single slash which directed ODT to reopen the last location opened. This allowed us to verify that the word 002340 was correctly stored in location 1000.

Note again, that opening a location while another is currently open will automatically close the currently open location before opening the new location.

Also note that if you specify the opening of an odd numbered address with a slash, ODT will open the location as a byte, and subsequently will behave as if a backslash had been typed.

2.2.2 The Backslash, \

In addition to operating on words, ODT-11R operates on bytes. One way to open a byte is to type the address of the byte followed by a backslash. (\ is printed by typing SHIFT/L). This not only causes the byte value at the specified address to be printed out, it also causes the value to be interpreted as ASCII code, and the corresponding character to be echoed (if possible) on the teleprinter.

```
*1001\101=A
```

A backslash typed alone will reopen the last open byte. If a word was previously open, the backslash will reopen its even byte.

```
*1002/000004\004=
```

The LINE FEED and up-arrow (or circumflex) keys will operate on bytes if a byte is open when the command is given (see Sections 2.2.3, and 2.2.4). For example:

```
*1001\101=A↓
001002\004=↓
001001\101=A
*
```

2.2.3 The LINE FEED Key, ↓

If the LINE FEED key is typed when a location is open, ODT closes the open location and opens the next sequential location:

```
*1000/002340↓     (↓ denotes typing the LINE FEED key)
001002/012740
```

In this example, the LINE FEED key instructed ODT to print the address of the next location along with its contents, and to wait for further instructions. After the above operation, location 1000 is closed and 1002 is open. The open location may be modified by typing the new contents.

If the opened location was a byte, the LINE FEED opens the next byte.

2.2.4 The Up-Arrow, ↑

The up-arrow (or circumflex) symbol is effected by typing the SHIFT and N key combination. If the up-arrow is typed when a location is open, ODT closes the open location and opens the previous location (as shown by continuing from the example above):

```
001002/012740↑     (↑ is printed by typing SHIFT and N)
001000/002340
```

Now location 1002 is closed and 1000 is open. The open location may be modified by typing the new contents.

If the opened location was a byte, then ↑ opens a byte as well.

2.2.5 The Back-Arrow, ←

The back-arrow (or underline) symbol is effected by typing the SHIFT and O key combination. If the back-arrow is typed to an open word, ODT interprets the contents of the currently open word as an address indexed by the Program Counter (PC) and opens the location so addressed:

```
*1006/000006←     (←  is printed by typing SHIFT and O)
001016/100405
```

Notice in this example that the open location, 1006, was indexed by the PC as if it were the operand of an instruction with address mode 67 as explained in the PAL-11R Assembler Programmer's Manual.

A modification to the opened location can be made before a ↓, ↑, or ←, is typed. Also, the new contents of the location will be used for address calculations using the ← command. Example:

```
*100/000222    4↓     (modify to 4 and open next location)
000102/000111  6↑     (modify to 6 and open previous location)
000100/000004   100←  (change to 100 and open location indexed
000202/(contents)      by PC)
```

2.2.6 Open the Addressed Location, @

The symbol @ will optionally modify, close an open word, and use its contents as the address of the location to open next.

```
*1006/001024   @       (open location 1024 next)
001024/000500
*1006/001024   2100    @(modify to 2100 and open
002100/177774            location 2100)
```

2.2.7 Relative Branch Offset >

The right angle bracket, >, will optionally modify, close an open word, and use its low-order byte as a relative branch offset to the next word opened.

```
*1032/000407  301>      (modify to 301 and interpret
000636/000010            as a relative branch)
```

2.2.8 Return to Previous Sequence, <

The left-angle bracket, <, will optionally modify, close an open location, and open the next location of the previous sequence interrupted by a ←, @, or > command. Note that ←, @, or > will cause a sequence change to the word opened. If a sequence change has not occurred, < will simply open the next location as a LINE FEED does. The command will operate on both words and bytes.

```
*1032/000407     301 >   (> causes a sequence change)
000636/000010 <          (< causes a return to original
001034/001040 @          (sequence)
                         (@ causes a sequence change)
001040/000405\005=   <   (< now operates on byte)
001035  \002=  <         (< acts like ↓)
001036  \004=
```

2.2.9 Accessing General Registers 0-7

The program's general registers 0-7 can be opened using the following command format:

```
*$n/
```

where n is the integer representing the desired register (in the range 0 through 7). When opened, these registers can be examined or changed by typing in new data as with any addressable location. For example:

```
*$0/000033               (R0 was examined and closed)
*
and
*$4/000474  464          (R4 was opened, changed, and
*                         closed)
```

The example above can be verified by typing a slash in response to ODT's asterisk:

```
*/000464
```

The ↓, ↑, ← or @ commands may be used when a register is open.

2.2.10 Acessing Internal Registers

The program's Status Register contains the condition codes of the most recent operational results and the interrupt priority level of the object program. It is opened using the following command:

```
*$S/000311
```

where $S represents the address of the Status Register. In response to $S in the example above, ODT printed the 16-bit word of which only the low-order 8 bits are meaningful: Bits 0-3 indicate whether

a carry, overflow, zero, or negative (in that order) has resulted, and bits 5-7 indicate the interrupt priority level (in the range 0-7) of the object program. (See the PDP-11 Handbook for the Status Register format.)

The $ is used to open certain other internal locations:

$B location of the first word of the breakpoint table (see Section 2.3).

$M mask location for specifying which bits are to be examined during a bit pattern search (see Section 2.6).

$P location defining the operating priority of ODT (see Section 2.12).

$S location containing the condition codes (bits 0-3) and interrupt priority level (bits 5-7).

$C location of the Constant Register (see Section 2.7).

$R location of Relocation Register 0, the base of the Relocation Register table (see Section 2.10).

$F location of Format Register (see Section 2.1).

2.2.11 Radix 50 Mode,X

The Radix 50 mode of packing certain ASCII characters three to a word is employed by many DEC-supplied PDP-11 system programs, and may be employed by any programmer via the Assembler's ".RAD50" directive.

ODT-11R provides a method for examining and changing memory words packed in this way with the "X" command.

When a word is opened, the user may type "X", in which case ODT will convert the contents of the opened word to its 3-character Radix 50 equivalent, and will type these characters on the Teletype.

You may then type one of the following:

	Type	Effect
a.	RETURN key	closes the currently open location
b.	LINE FEED key	closes the location and opens the next one in sequence

c.	↑ key	closes the location and opens the previous one in sequence.
d.	Any three characters whose octal code is 040 (space) or greater.	convert the three specified characters into packed Radix 50 format.

Legal Radix 50 characters are:

$	Space	0 through 9	A through Z

If any other characters are typed, the resulting binary number is unspecified. However, exactly three characters must be typed before ODT resumes its normal mode of operation.

After the third character is typed, the resulting binary number is available to be stored into the opened location by closing the location in any one of the usual ways (carriage-return, line feed, etc.). Example:

```
*1000/042431    X=KBI  CBA
*1000/011421    X=CBA
```

WARNING

After ODT has converted the three characters to binary, the binary number can be interpreted in one of many different ways, depending on the command which follows. For example:

```
*1234/063337  X=PRO XIT/
```

Since the Radix 50 equivalent of XIT is 113574, the final slash typed in the example will cause ODT to open location 113574 if it is a legal address. (See Chapter 3 for a discussion of command legality and detection of errors.)

2.3 BREAKPOINTS

The breakpoint feature facilitates monitoring the progress of program execution. A breakpoint may be set at any instruction which is not referenced by the program for data. When a breakpoint is set, ODT replaces the contents of the breakpoint location with a trap instruction so that when the program is executed and the

breakpoint is encountered, program execution is suspended, the original contents of the breakpoint location are restored, and ODT regains control.

With ODT-11R you can, at any one time, have up to eight breakpoints set, numbered 0 through 7. The r;B command will set the next available breakpoint. Specific breakpoints may be set or changed by the r;nB command where n is the number of the breakpoint. For example:

```
*1020;B              (sets breakpoint 0)
*1030;B              (sets breakpoint 1)
*1040;B              (sets breakpoint 2)
*1032;1B             (resets breakpoint 1)
*
```

The ;B command removes all breakpoints. To remove only one of the breakpoints the ;nB command is used, where n is the number of the breakpoint. For example:

```
*;2B                 (removes the second breakpoint)
*
```

The $B/ command opens the location containing the address of breakpoint 0. The next seven locations contain the addresses of the other breakpoints in order, and thus can be opened using the LINE FEED key. (The next location is for Single-instruction mode, explained in Section 2.5). Example:

```
*$B/001020  ↓
nnnnnn/001032↓
nnnnnn/ (address internal to ODT)
```

In this example, breakpoint 2 is not set. The contents will be an address internal to ODT. After the table of breakpoints is the table of Proceed command repeat counts, first for each breakpoint, and then for the Single Instruction mode (see Section 2.5).

```
.
.
.                ↓
nnnnnn/001036    ↓     (breakpoint 7)
nnnnnn/nnnnnn    ↓     (single-instruction address)
nnnnnn/000000   15 ↓   (count for breakpoint 0)
nnnnnn/000000          (count for breakpoint 1)
```

2.4 RUNNING THE PROGRAM, r;G AND r;P

Program execution is under control of ODT. There are two commands for running the program: r;G and r;P. The r;G command is used to start execution (Go) and r;P to continue (Proceed) execution after having halted at a breakpoint. For example:

```
*1000;G
```

starts execution at location 1000. The program will run until encountering a breakpoint or until program completion, unless it gets caught in an infinite loop, where you must either restart or reenter as explained in Section 5.2.

When a breakpoint is encountered, execution stops and ODT prints Bn; (where n is the breakpoint number), followed by the address of the breakpoint. You may then examine desired locations for expected data. For example:

```
*1010;3B            (breakpoint 3 is set at location
                     1010)
*1000;G             (execution started at location 1000)
*B3;001010          (execution stopped at location 1010)
*
```

When a breakpoint is set in a loop, it may be desirable to allow the program to execute a certain number of times through the loop before recognizing the breakpoint. This may be done by typing the k;P command and specifying the number of times the breakpoint is to be encountered before program execution is suspended (on the kth encounter).

The count, k, is associated only with the numbered breakpoint which most recently occurred. A different proceed count may be associated with each numbered breakpoint, and will apply to that breakpoint only. Example:

```
B3;001010       (execution halted at breakpoint)
*1250;B         (set breakpoint at location 1250)
*4;P            (continue execution, loop through
B3;001250       breakpoint 3 times and halt on 4th
*               occurrence of the breakpoint)
```

The breakpoint repeat counts can be inspected by typing $B/ and following that with the typing of nine LINE FEEDs. The repeat count for breakpoint 0 will then be printed. The repeat counts for breakpoints 1 through 7, and the repeat count for the single instruction trap follow in sequence (see Section 2.5). Opening any one of these provides an alternative way of specifying the count. The location, being open, can have its contents modified in the usual manner by the typing of new contents and then the RETURN key.

Breakpoints are inserted when performing an r;G or k;P command. Upon execution of the r;G or k;P command, the general registers 0-6 are set to the values in the locations specified as $0-$6 and the processor status register is set to the value in the location specified as $S.

2.5 SINGLE-INSTRUCTION MODE

With this mode you can specify the number of instructions you wish executed before suspension of the program run. The Proceed command, instead of specifying a repeat count for a breakpoint encounter, specifies the number of succeeding instructions to be executed. Note that breakpoints are disabled when single-instruction mode is operative.

Commands for single-instruction mode follow:

;nS	Enables Single-instruction mode (n can have any value and serves only to distinguish this form from the form ;S); breakpoints are disabled.
k;P	Proceeds with program run for next k instructions before reentering ODT (if k is missing, it is assumed to be 1). (Trap instructions and associated handlers can affect the Proceed repeat count. See Section 4.2)
;S	Disables Single-instruction mode.

When the repeat count for Single-instruction mode is exhausted and the program suspends execution, ODT prints:

```
B8;k
*
```

where k is the address of the next instruction to be executed. The $B breakpoint table contains this address following that of breakpoint 7. However, unlike the table entries for breakpoints 0-7, direct modification has no effect.

Similarly, following the repeat count for breakpoint 7, is the repeat count for single-instruction mode. This table entry, though, may be directly modified, and thus is an alternative way of setting the Single-instruction mode repeat count. In such a case, ;P implies the argument set in the $B repeat count table rather than 1.

2.6 SEARCHES

With ODT you can search all or any specified portion of core memory for any specific bit pattern or for references to a specific location.

2.6.1 Word Search, r;W

Before initiating a word search, the mask and search limits must be specified as shown in the example below. The location represented by $M is used to specify the mask of the search. $M/ opens the mask register. The next two sequential locations (opened by LINE FEEDs)contain the lower and upper limits of the search. Bits set to 1 in the mask will be examined during the search; other bits will be ignored. Then the search object and the initiating command are given using the r;W command where r is the search object. When a match is found the address of the unmasked matching word is printed. For example:

```
*$M/000000  177400 ↓    (test high order eight bits)
nnnnnn/000000  1000 ↓   (set low address limit)
nnnnnn/000000  1040     (set high address limit)
*400;W                  (initiate word search)
001010/000770
001034/000404
*
```

In the search process, the word currently being examined and the search object are exclusive ORed (XORed), and the result is ANDed to the mask. If this result is zero, a match has been found, and is reported on the teleprinter. Note that if the mask is zero, all locations within the limits will be printed.

Typing CTRL/U during a search printout will terminate the search.

2.6.2 Effective Address Search, r;E

ODT enables you to search for words which address a specified location. Open the mask register only to gain access to the low- and high-limit registers. After specifying the search limits (Section 2.6.1) the command r;E is typed (where r is the effective address), initiating the search.

Words which are either an absolute address (argument n itself) a relative address offset, or a relative branch to the effective address will be printed after their addresses. For example:

```
*$M/177400↓
nnnnnn/001000  1010↓     (open mask register only to gain
nnnnnn/001040  1060       access to search limits)
*1034;E                  (initiating search)
001016/001006            (relative branch)
001054/002767            (relative branch)
*1020;E                  (initiating a new search)
001022/177774            (relative address offset)
001030/001020            (absolute address)
*
```

Particular attention should be given to the reported references to the effective address because a word may have the specified bit pattern of an effective address without actually being so used. ODT will report these as well.

Typing CTRL/U during a search printout will terminate the search.

2.7 THE CONSTANT REGISTER, r;C

It is often desirable to convert a relocatable address into a relocated address or to convert a number into its two's complement, and then to store the converted value into one or more places in your program. The Constant Register provides a means of accomplishing this and other useful functions.

When r;C is typed, the relocatable expression r is evaluated to its six digit octal value and is both printed on the teleprinter

and stored in the Constant Register. The contents of the Constant Register may be invoked in subsequent relocatable expressions by typing the letter C.

Examples:

*-4432;C=173346	(The two's complement of 4432 is placed in the Constant Register)
*1000/001000 C	(The contents of the Constant Register are stored in location 1000)
*1000;1R	(Relocation Register 1 is set to 1000)
*1,4272;C=005272	(Relative location 4272 is reprinted as an absolute location and stored in the Constant Register)

2.8 CORE BLOCK INITIALIZATION, ;F AND ;I

The Constant Register can be used in conjunction with the commands ;F and ;I to set a block of memory to a given value. While the most common value required is zero, other possibilities are plus one, minus one, ASCII space, etc.

When the command ;F is typed, ODT-11R stores the contents of the Constant Register in successive memory words starting at the memory word address specified in the lower search limit, and ending with the address specified in the upper search limit.

When the command ;I is typed, the low-order 8 bits in the Constant Register are stored in successive bytes of memory starting at the byte address specified in the lower search limit and ending with the byte address specified in the upper search limit.

Example: Assume relocation register 1 contains 1000, 2 contains 2000, and 3 contains 3000. The following sequence sets word locations 1000-1776 to zero, and byte locations 2000-2777 to ASCII spaces.

```
*$M/000000↓                (open mask register to gain access
                            to search limits)
nnnnnn/000000  1,0↓        (sets lower limit to 1000)
nnnnnn/000000  2,-2        (sets upper limit to 1776)
*0;C=000000                (Constant Register set to zero)
*;F                        (Locations 1000-1776 set to zero)
*$M/000000↓
nnnnnn/001000  2,0↓        (Sets lower limit to 2000)
nnnnnn/001776  3,-1        (Sets upper limit to 2777)
*40;C=000040               (Constant Register set to 40
                            (SPACE) )
*;I                        (Byte locations 2000-2777 are set
*                          to value in low order 8 bits of
                           Constant Register)
```

2.9 CALCULATING OFFSETS, r;O

Relative addressing and branching involve the use of an offset - the number of words or bytes forward or backward from the current location to the effective address. During the debugging session it may be necessary to change a relative address or branch reference by replacing one instruction offset with another. ODT calculates the offsets for you in response to its r;O command.

The command r;O causes ODT to print the 16-bit and 8-bit offsets from the currently open location to address r. For example:

```
*346/000034  414;O  000044  022  22
*/000022
```

In the example, location 346 is opened and the offsets from that location to location 414 are calculated and printed. The contents of location 346 are then changed to 22 (the 8-bit offset) and verified on the next line.

The 8-bit offset is printed only if it is in the range -128_{10} to 127_{10} and the 16-bit offset is even, as was the case above. For example, the offset of a relative branch is calculated and modified as follows:

```
*1034/103421  1034;O  177776  377\021 377
*103777
```

Note that the modified low-order byte 377 must be combined with the unmodified high-order byte.

2.10 RELOCATION REGISTER COMMANDS, r;nR, ;nR, ;R

The use of the relocation registers has been defined in Section 1.1. At the beginning of a debugging session it will be desirable to preset the registers to the relocation biases of those relocatable modules which will be receiving the most attention.

This can be done by typing the relocating bias, followed by a semicolon and the specification of relocation registers.

```
r;nR
```

r may be any relocatable expression and n is an integer from 0 to 7. If n is omitted it is assumed to be 0.

As an example:

```
*1000;5R                    (puts 1000 into relocation
                             register 5)
*5,100;5R                   (effectively adds 100 to the
                             contents of relocation register
                             5)
*
```

In certain uses programs may be relocated to an address below that at which they were assembled. This could occur with PIC code which is moved without the use of the Linker.

In this case the appropriate relocation bias would be the 2's complement of the actual downward displacement. One method for easily evaluating the bias and putting it in the relocation register is illustrated in the following example:

Suppose the program was assembled at location 5000 and was moved to location 1000. Then the sequence:

```
*1000;1R
*1,-5000;1R
*
```

puts the 2's complement of 4000 in relocation register 1, as desired.

Relocation registers are initialized to -1, so that unwanted relocation registers will never enter into the selection process when ODT searches for the most appropriate register.

To set a relocation register to -1, type ;nR. To set all relocation registers to -1, type ;R.

ODT maintains a table of relocation registers, beginning at the address specified by $R. Opening $R ($R/) opens relocation register Ø. Successively typing the LINE FEED key opens the other relocation registers in sequence. When a relocation register is opened in this way, it may be modified just as any other memory location.

2.11 THE RELOCATION CALCULATORS, nR AND n!

When a location has been opened, it is often desirable to relate the relocated address and the contents of the location back to their relocatable values. To calculate the relocatable address of the opened location relative to a particular relocation bias, type n!, where n specifies the relocation register. This calculator works with opened bytes and words. If n is omitted, the relocation register whose contents are closest but less than or equal to the opened location is selected automatically by ODT. In the following example, assume that these conditions are fulfilled by relocation register 2, which contains 2000:

To find the most likely module that a given opened byte is in,

```
*2500\011=  !=2,000500
```

Typing nR after opening a word causes ODT-11R to print the octal number which equals the value of the contents of the opened location minus the contents of relocation register n. If n is omitted, ODT-11R selects the relocation register whose contents are closest but less than or equal to the contents of the opened location. For example, assume the relocation bias stored in relocation register 1 is 001234; then:

```
*1,500/024550    1R=1,023314
```

The value 23314 is the content of 1,500, relative to the base 1234. An example of the use of both:

If relocation register 1 contains 1000, and relocation register 2 contains 2000, then to calculate the relocatable addresses of location 3000 and its content, relative to 1000 and 2000, the following can be performed.

```
*3000/005670 1!=1,002000 2!=2,001000 1R=1,004670 2R=2,003670
```

2.12 ODT's PRIORITY LEVEL, $P

$P represents a location in ODT that contains the priority level at which ODT operates. If $P contains the value 377, ODT will operate at the priority level of the processor at the time ODT is entered. Otherwise $P may contain a value between 0 and 7 corresponding to the fixed priority at which ODT will operate.

To set ODT to the desired priority level, open $P. ODT will print the present contents, which may then be changed:

```
*$P/000006  377
*
```

If $P is not specified, its value will be seven.

Breakpoints may be set in routines at different priority levels. For example, a program running at a low priority level may use a device service routine which operates at a higher priority level. If a breakpoint occurs from a low-priority routine, if ODT operates at a low priority, and if an interrupt does occur from a high priority routine, then the breakpoints in the high priority routine will not be executed since they have been removed.

2.13 ASCII Input and Output, r;nA

ASCII text may be inspected and changed by the command

r;nA

where r is a relocatable expression, and n is a character count. If n is omitted it is assumed to be 1. ODT prints n characters starting at location r, followed by a <CR><LF>. You may then type one of the following:

a. <CR> ODT outputs <CR><LF><*> and waits for another command.

b. <LF> ODT opens the byte following the last byte output.

c. <up to n characters of text>

ODT inserts the text into core, starting at location r.

If less than n characters are typed, you must terminate the command by typing CTRL/U, causing <CR><LF><*> to be output as in case a. above. However, if exactly n characters are typed, ODT responds with:

<CR> <LF> <address of next available byte> <CR> <LF> <*>

Note that n may actually be expressed as a relocatable expression, and could be quite large, accidentally. There is no safeguard against this in ODT11R.

2.14 Return to Monitor, CTRL/C

If ODT is awaiting a command, a CTRL/C from the keyboard will simulate a TTY interrupt and effectively wake up the Monitor. The Monitor responds with a ↑C on the teleprinter and returns to ODT at a wait loop (BR .). See Chapter 3 of the DOS Monitor Programmer's Handbook for a more detailed description of CTRL/C.

CHAPTER 3

ERROR DETECTION

ODT-11R informs you of two types of error: illegal or unrecognizable command and bad breakpoint entry.

ODT-11R does not check for the legality of an address when commanded to open a location for examination or modification.

Thus the command:

```
177774/
```

will reference nonexistent memory, thereby causing a trap through the vector at location 4. If this vector has not been properly initialized, unpredictable results will occur.

Similarly, a command such as

```
$20/
```

which references an address eight times the value represented by $2, may cause an illegal (nonexistent) memory reference.

Typing something other than a legal command will cause ODT to ignore the command, print

```
*  ?
```

and wait for another command. Therefore, to cause ODT to ignore a command just typed, type any illegal character (such as 9 or RUBOUT) and the command will be treated as an error, i.e., ignored.

ODT suspends program execution whenever it encounters a breakpoint, i.e., a trap to its breakpoint routine. If the breakpoint routine is entered and no known breakpoint caused the entry, ODT prints:

```
BE001542
*
```

and waits for another command. In the example above, BE001542 denotes Bad Entry from location 001542. A bad entry may be caused by an illegal trace trap instruction, setting the T-bit in the status register, or by a jump to the middle of ODT.

CHAPTER 4

PROGRAMMING CONSIDERATIONS

Information in this section is not necessary for the efficient use of ODT. However, its content does provide a better understanding of how ODT performs some of its functions.

4.1 FUNCTIONAL ORGANIZATION

The internal organization of ODT is almost totally modularized into independent subroutines. The internal structure consists of three major functions: command decoding, command execution, and various utility routines.

The command decoder interprets the individual commands, checks for command errors, saves input parameters for use in command execution, and sends control to the appropriate command execution routine.

The command execution routines take parameters to be saved by the command decoder and use the utility routines to execute the specified command. Command execution routines exit either to the object program or back to the command decoder.

The utility routines are common routines such as SAVE-RESTORE and I/O. They are used by both the command decoder and the command executers. (See Figure 1.)

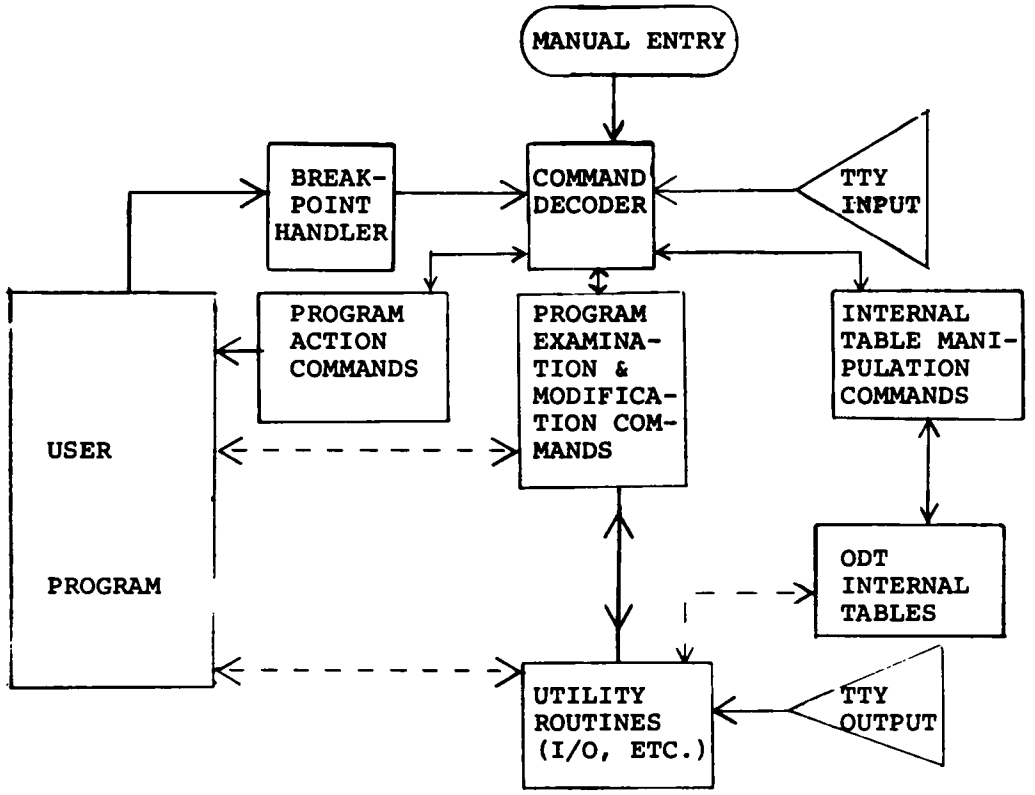
4.2 BREAKPOINTS

The function of a breakpoint is to give control to ODT whenever the user program tries to execute the instruction at the selected address. Upon encountering a breakpoint, the user can utilize all of the ODT commands to examine and modify his program.

When a breakpoint is executed, ODT-11R removes all the breakpoint instructions from the user's code so that the locations may be examined and/or altered. ODT then types a message to the user

FIGURE 1

COMMUNICATION AND DATA FLOW



of the form Bn;k where k is the breakpoint address (and n is the breakpoint number). The breakpoints are automatically restored when execution is resumed.

A major restriction in the use of breakpoints is that the word where a breakpoint has been set must not be referenced by the program in any way since ODT has altered the word. Also, no breakpoint should be set at the location of any instruction that clears the T-bit. For example:

```
MOV #240,177776          ;SET PRIORITY TO LEVEL 5
```

> Note that instructions that cause or return from traps (e.g., EMT, RTI) are likely to clear the T-bit, since a new word from the trap vector or the stack will be loaded into the Status Register.

A breakpoint occurs when a trace trap instruction (placed in the user program by ODT) is executed. When a breakpoint occurs, the following steps are taken:

1. Set processor priority to seven (automatically set by trap instruction).
2. Save registers and set up stack.
3. If internal T-bit trap flag is set, go to step 13.
4. Remove breakpoints.
5. Reset processor priority to ODT's priority or user's priority.
6. Make sure a breakpoint or Single-instruction mode caused the interrupt.
7. If the breakpoint did not cause the interrupt, go to step 15.
8. Decrement repeat count.
9. Go to step 18 if non-zero; otherwise reset count to one.
10. Save Teletype status.
11. Type message to user about the breakpoint or Single-instruction mode interrupt.
12. Go to command decoder.
13. Clear T-bit in stack and internal T-bit flag.
14. Jump to the Go processor.
15. Save Teletype status.

16. Type BE (Bad Entry) followed by the address.

17. Clear the T-bit, if set, in the user status and proceed to the command decoder.

18. Go to the Proceed processor, bypassing the TTY restore routine.

Note that steps 1-5 inclusive take approximately 100 microseconds during which time interrupts are not permitted to occur (ODT is running at level 7).

When a proceed (;P) command is given, the following occurs:

1. The proceed is checked for legality.

2. The processor priority is set to seven.

3. The T-bit flags (internal and user status) are set.

4. The user registers, status, and Program Counter are restored.

5. Control is returned to the user.

6. When the T-bit trap occurs, steps 1, 2, 3, 13, and 14 of the breakpoint sequence are executed, breakpoints are restored, and program execution resumes normally.

When a breakpoint is placed on an IOT, EMT, TRAP, or any instruction causing a trap, the following occurs:

1. When the breakpoint occurs as described above, ODT is entered.

2. When ;P is typed, the T-bit is set and the IOT, EMT, TRAP, or other trapping instruction is executed.

3. This causes the current PC and status (with the T-bit included) to be pushed on the stack.

4. The new PC and status (no T-bit set) are obtained from the respective trap vector.

5. The whole trap service routine is executed without any breakpoints.

6. When an RTI is executed, the saved PC and PS (including the T-bit) are restored. The instruction following the trap-causing instruction is executed. If this instruction is not another trap-causing instruction, the T-bit trap occurs, causing the breakpoints to be reinserted

in the user program, or the Single-instruction mode repeat count to be decremented. If the following instruction is a trap-causing instruction, this sequence is repeated starting at step 3.

NOTE

Exit from the trap handler must be via the RTI instruction. Otherwise, the T-bit will be lost. ODT will not gain control again since the breakpoints have not yet been reinserted.

Note that the ;P command is illegal if a breakpoint has not occurred (ODT will respond with ?); ;P is legal, however, after any trace trap entry.

The internal breakpoint status words have the following format:

1. The first eight words contain the breakpoint addresses for breakpoints 0-7. (The ninth word contains the address of the next instruction to be executed in Single-instruction mode.)

2. The next eight words contain the respective repeat counts. (The following word contains the repeat count for Single-instruction mode.)

These words may be changed at will by the user, either by using the breakpoint commands or by direct manipulation with $B.

When program runaway occurs (that is, when the program is no longer under ODT control, perhaps executing an unexpected part of the program where a breakpoint has not been placed) ODT may be given control by pressing the HALT key to stop the machine, and restarting ODT (see Section 5.2). ODT will print *, indicating that it is ready to accept a command.

If the program being debugged uses the teleprinter for input or output, the program may interact with ODT to cause an error since ODT uses the teleprinter as well. This interactive error will not occur when the program being debugged is run without ODT.

1. If the teleprinter interrupt is enabled upon entry to the ODT break routine, and no output interrupt is pending when ODT is entered, ODT will generate an unexpected interrupt when returning control to the program.

2. If the interrupt of the teleprinter reader (the keyboard) is enabled upon entry to the ODT break routine, and the program is expecting to receive an interrupt to input a character, both the expected interrupt and the character will be lost.

3. If the teleprinter reader (keyboard) has just read a character into the reader data buffer when the ODT break routine is entered, the expected character in the reader data buffer will be lost.

4.3 SEARCH

The word search allows the user to search for bit patterns in specified sections of memory. Using the $M/ command, the user specifies a mask, a lower search limit ($M+2), and an upper search limit ($M+4). The search object is specified in the search command itself.

The word search compares selected bits (where ones appear in the mask) in the word and search object. If all of the selected bits are equal, the unmasked word is printed.

The search algorithm is:

1. Fetch a word at the current address.
2. XOR (exclusive OR) the word and search object.
3. AND the result of step 2 with the mask.
4. If the result of step 3 is zero, type the address of the unmasked word and its contents. Otherwise, proceed to step 5.
5. Add two to the current address. If the current address is greater than the upper limit, type * and return to the command decoder, otherwise go to step 1.

Note that if the mask is zero, ODT will print every word between the limits, since a match occurs every time (i.e., the result of step 3 is always zero).

In the effective address search, ODT interprets every word in the search range as an instruction which is interrogated for a possible direct relationship to the search object. The mask register is opened only to gain access to the search limit registers.

The algorithm for the effective address search is (where (X) denotes contents of X, and K denotes the search object):

1. Fetch a word at the current address X.
2. If (X)=K [direct reference], print contents and go to step 5.
3. If (X)+X+2=K [indexed by PC], print contents and go to step 5.
4. If (X) is a relative branch to K, print contents.
5. Add two to the current address. If the current address is greater than the upper limit, perform a carriage return/line feed and return to the command decoder; otherwise, go to step 1.

4.4 TELETYPE*INTERRUPT

Upon entering the TTY SAVE routine, the following occurs:

1. Save the LSR status register (TKS).
2. Clear interrupt enable and maintenance bits in the TKS.
3. Save the TTY status register (TPS).
4. Clear interrupt enable and maintenance bits in the TPS.

To restore the TTY:

1. Wait for completion of any I/O from ODT.
2. Restore the TKS.
3. Restore the TPS.

*Teletype is a registered trademark of the Teletype Corporation.

WARNINGS

If the TTY printer interrupt is enabled upon entry to the ODT break routine, the following may occur:

1. If no output interrupt is pending when ODT is entered, an additional interrupt will always occur when ODT returns control to the user.

2. If an output interrupt is pending upon entry, the expected interrupt will occur when the user regains control.

If the TTY reader (keyboard) is busy or done, the expected character in the reader data buffer will be lost.

If the TTY reader (keyboard) interrupt is enabled upon entry to the ODT break routine, and a character is pending, the interrupt (as well as the character) will be lost.

CHAPTER 5

OPERATING PROCEDURES

This section describes loading procedures for ODT, restarting and reentering procedures, and error recovery.

5.1 LOADING PROCEDURES

ODT-11R is supplied as a relocatable object module. It should be linked with other object modules which form the program to be debugged. The resultant load module is loaded into core memory using the System Loader, as explained in the Disk Operating System Monitor Programmer's Handbook.

5.2 STARTING AND RESTARTING

After loading the load module (including ODT) into core via the Monitor GET command, ODT may be started by means of the Monitor command OD. ODT indicates its readiness to accept input by printing:

```
ODT11R Vnnnn          (nnnn is the ODT version number)
*
```

When ODT is started at its start address, the SP register is set to an ODT internal stack, registers R0-R6 are saved in $0-$6, the teleprinter status and Central Processor status are saved, and the trace trap vector is initialized. If ODT is started at its start address after breakpoints have been set in a program, ODT will forget about the breakpoints and will leave the program modified, i.e., the breakpoint instructions will be left in the program.

ODT's start address is determined from the Linker's load map, i.e., the low limit address plus 172_8 equals ODT's start address.

There are two ways of restarting ODT:

1. Restart at start address + 2 (use the DOS command OD R)
2. Reenter at start address + 4 (use the DOS command OD K)

To restart, use the Monitor command OD R. A restart will save the general registers, clear the relocation registers, remove the breakpoint instructions from the user program, and then forget all breakpoints, i.e., simulate the ;B command.

To reenter, use the Monitor command OD K. A reenter will save the Processor Status and general registers, remove the breakpoint instructions from the user program, and ODT will type the BE (Bad Entry) error message. ODT will remember which breakpoints were set and will reset them on the next ;G command (;P is illegal after a Bad Entry).

INDEX

Absolute address, 1-2, 2-1
Accessing general
  registers 0-7, 2-6
Address
  absolute, 2-1
  relative, 2-1
  search, 2-13, 4-7
Algorithm
  effective address search, 4-7
  word search, 4-6
ANDing, 2-13, 4-6
ASCII
  character packing, 2-7
  code, 2-3
  I/O, 2-19
Asterisk symbol (*) usage, 2-1
At symbol (@) usage, 2-5
Automatic relocation facility, 1-2

Back-arrow symbol (←) usage, 2-4
Backslash symbol (\ ) usage, 2-3
Bad breakpoint entry, 3-1
Base address, 1-2
Branching, 2-15
Breakpoint, 2-8, 2-9, 4-1
  priority levels, 2-18
  removing, 2-9
  repeat count, 2-11, 2-12
  restoration, 4-3
  setting, 2-9
  status words, 4-5
Byte opening, 2-3

Calculating offsets, 2-15
Changing location, 2-2
Circumflex symbol (^) usage, 1-4, 2-3
Closing location, 2-2, 2-3
Command
  decoder, 4-1
  execution, 4-1
  illegal, 3-1
Commands, 1-4, 2-1
  list of, 1-4, -5, -6
Communication and data flow
  chart, 4-2
Condition codes, 2-6
Constant register, 1-3, 2-13
Conversion
  number, 2-13
  relocatable address, 2-13
Core block initialization, 2-14
Count specification, 2-11

Description of ODT-11R, 1-1
Dollar sign symbol ($) usage, 2-6, 2-7

Effective address search, 4-7
Error caused by teleprinter
  I/O, 4-5
Error detection, 3-1
Exclamation point (!) usage, 2-17
Exclusive OR, 2-13, 4-6
Exit from trap handler, 4-5
Expressions, relocatable, 1-2

Flow chart: Communication and
  Data, 4-2
Format register ($F), 2-2

HALT key, 4-5

Illegal command, 3-1
Infinite looping, 2-10
Internal location opening, 2-7
Internal organization, 4-1
Interrupt priority level, 2-6, 2-7

Left angle bracket (<) usage, 2-5
Limits, search, 2-12
LINE FEED key, 2-3, 2-4
Loading procedures, 5-1
Location
  opening, changing, closing, 2-2
  priority level, 2-8
Looping, 2-10
  infinite, 2-10

Mask, 2-12
Minus sign usage, 1-2
Mode X, 2-7
Modifying open location, 2-5

Nonexistent memory reference, 3-1

Offset calculation, 2-15
Open location, 2-2, 2-3
Opening
  addressed location, 2-5
  byte, 2-3, 2-4
  internal locations, 2-7
Operating procedures, 5-1
Organization, internal, 4-1

Priority level, 2-18
Proceed command, 4-4
Proceed count, 2-10
Program counter (PC), 2-4
Program execution suspension, 2-9
Program runaway, 4-5

Radix 50 characters, 2-8
Radix 50, Mode X, 2-7
Reentering, 5-2
Relative
  address, 2-1, 2-2, 2-15
  branch offset, 2-5
Relocatable
  code, 1-2
  expressions, 1-2, 1-3
Relocated modules, 1-2
Relocation, 1-2
  bias, 1-2, 2-17
  calculators, 2-17
  facility, automatic, 1-2
  registers, 1-2
  register table, 2-17
Removing breakpoints, 2-9
Repeat counts, 2-11, 2-12
Restarting, 5-1, 5-2
Restoring breakpoints, 4-3
Return to previous sequence, 2-5
Right angle bracket (>) usage,2-5
Runaway program, 4-5
Running the program, 2-10

Search,
  effective address, 2-13, 4-7
  termination, 2-13
  word, 2-12, 4-6
Setting breakpoints, 2-9
Single-instruction mode, 2-11
Slash (/) usage, 2-2, 2-6
Starting and restarting, 5-1
Status register, 2-6
Suspension of program execution, 2-9

Table
  breakpoint, 2-9
  Proceed command repeat counts, 2-9
  relocation registers, 2-17
T-bit, 4-3
Teleprinter I/O error, 4-5
Termination of search, 2-13
Trace trap instruction, 4-3
Trap handler exit, 4-5
Trap instruction, 2-8, 4-3
TTY SAVE routine, 4-7
Two's complement, 1-2

Underline symbol (_) usage, 1-4, 2-4
Up-arrow symbol (↑) usage, 2-3, 2-4
Utility routines, 4-1

XOR (exclusive OR), 2-13, 4-6

Word search, 2-12, 4-6

PDP-11

FILE UTILITY PACKAGE

(PIP)

FOR THE

DISK OPERATING SYSTEM

SOFTWARE SUPPORT CATEGORY

The software described in this document is supported by DEC under Category I, as defined on page iv of this document.

For additional copies, order No. DEC-11-PIDB-D from Digital Equipment Corporation, Direct Mail, Building 1-1, Maynard, Massachusetts 01754

First Printing, May 1971
Second Printing, September 1971

Your attention is invited to the last two pages of this document. The "How to Obtain Software Information" page tells you how to keep up-to-date with DEC's software. The "Reader's Comments" page, when filled in and mailed, is beneficial to both you and DEC; all comments received are considered when documenting subsequent manuals.

Copyright © 1971 by Digital Equipment Corporation

Associated PDP-11 Documents:

Disk Operating System Monitor,
Programmer's Handbook, DEC-11-MWDC-D

PAL-11R Assembler,
Programmer's Manual, DEC-11-ASDC-D

Edit-11 Text Editor,
Programmer's Manual, DEC-11-EEDA-D

ODT-11R Debugging Program,
Programmer's Manual, DEC-11-OODA-D

Link-11 Linker and Libr-11 Librarian,
Programmer's Manual, DEC-11-ZLDC-D

FORTRAN IV,
Programmer's Manual, DEC-11-KFDA-D

This document is for information purposes and is subject to change without notice.

The following are trademarks of Digital Equipment Corporation

DEC	PDP-11
DECTAPE	RSTS-11
DIGITAL (logo)	RSX-11
COMTEX-11	UNIBUS

P R E F A C E

This manual describes the features and operation of the PIP File Utility Package (VØØ4A) for the PDP-11 Disk Operating System (DOS). Thus, the manual assumes familiarity with the DOS Monitor (see PDP-11 Disk Operating System Monitor, Programmer's Handbook, DEC-11-MWDC-D).

The File Utility Package is called PIP (Peripheral Interchange Program) for compatibility with similar programs on other DEC systems. Features provided by PIP include file transfers from one device to another, file deletions, file renaming, and directory listings.

Appendix B, Helpful Hints, contains valuable information on possible solutions to various problems which may arise while becoming familiar with PIP.

In addition to the File Utility Package and the DOS Monitor, the PDP-11 Disk Operating System software includes:

PAL-11R Assembler
Edit-11 Text Editor
ODT-11R Debugging Program
Link-11 Linker
Libr-11 Librarian
FORTRAN IV

SOFTWARE SUPPORT CATEGORIES

Digital Equipment Corporation (DEC) makes available four categories of software. These categories reflect the types of support a customer may expect from DEC for a specified software product. DEC reserves the right to change the category of a software product at any time. The four categories are as follows:

CATEGORY I
Software Products Supported at no Charge

This classification includes current versions of monitors, programming languages, and support programs provided by DEC. DEC will provide installation (when applicable), advisory, and remedial support at no charge. These services are limited to original purchasers of DEC computer systems who have the requisite DEC equipment and software products.

At the option of DEC, a software product may be recategorized from Category I to Category II for a particular customer if the software product has been modified by the customer or a third party.

CATEGORY II
Software Products that Receive Support for a Fee

This category includes prior versions of Category I programs and all other programs available from DEC for which support is given. Programming assistance (additional support), as available, will be provided on these DEC programs and non-DEC programs when used in conjunction with these DEC programs and equipment supplied by DEC.

CATEGORY III
Pre-Release Software

DEC may elect to release certain software products to customers in order to facilitate final testing and/or customer familiarization. In this event, DEC will limit the use of such pre-release software to internal, non-competitive applications. Category III software is only supported by DEC where this support is consistent with evaluation of the software product. While DEC will be grateful for the reporting of any criticism and suggestions pertaining to a pre-release, there exists no commitment to respond to these reports.

CATEGORY IV
Non-Supported Software

This category includes all programs for which no support is given

C O N T E N T S

			Page
CHAPTER 1		INTRODUCTION	1-1
	1.1	CALLING AND EXITING	1-1
	1.2	RECOVERING FROM ERRORS	1-3
CHAPTER 2		COMMANDS	2-1
	2.1	COMMAND STRINGS	2-1
	2.1.1	Device Specification	2-2
	2.1.2	Filename Specification	2-3
	2.1.3	Filename Extension Specification	2-3
	2.1.4	User Identification Code Specification	2-3
	2.1.5	Switch Specification	2-3
	2.2	SWITCHES	2-3
	2.2.1	Preaction Switches	2-5
	2.2.2	Action Switches	2-5
	2.2.3	Qualifying Switches	2-5
CHAPTER 3		FILE HANDLING	3-1
	3.1	FILE TRANSFERS	3-1
	3.1.1	Combining Files	3-1
	3.1.2	Modes During Transfers	3-2
	3.1.3	Formatted Modes	3-3
	3.1.4	Transfers from Teleprinter	3-3
	3.1.5	Asterisk in Transfers	3-4
	3.1.6	Contiguous Files	3-4
	3.2	RENAMING FILES	3-5
	3.3	DELETING FILES	3-6
	3.4	PROTECTING FILES	3-7
	3.5	RECOVERING FILES	3-8
	3.6	ALLOCATING CONTIGUOUS FILES	3-8
CHAPTER 4		DIRECTORY HANDLING	4-1
	4.1	ENTERING USER IDENTIFICATION INTO DIRECTORY	4-1
	4.2	DIRECTORY LISTINGS	4-1
	4.2.1	Full Directories	4-2
	4.2.2	Brief Directories	4-6
	4.2.3	Free Blocks	4-8
	4.3	ASTERISK IN DIRECTORY LISTINGS	4-9
	4.4	DECTAPE AND RK11 DISK INITIALIZATION	4-9
	4.4.1	DECtape	4-9
	4.4.2	RK11 Disk	4-10

CHAPTER 5	ERROR MESSAGES	5-1
APPENDIX A	SUMMARIES	A-1
APPENDIX B	HELPFUL HINTS	B-1
APPENDIX C	MAGTAPE OPERATIONS	C-1
APPENDIX D	PROGRAMMING CAUTIONS	D-1

CHAPTER 1

INTRODUCTION

The File Utility Package performs useful file operations for the PDP-11 Disk Operating System (DOS). Examples of such operations are file transfers between devices, directory listings of disk or DECtape, deleting files, and renaming files. This program is named PIP (Peripheral Interchange Program) to be compatible with similar programs on other DEC systems.

The following list summarizes the major features provided by PIP.

1. Allocate a contiguous file on DECtape or disk.
2. List the directory of a disk or DECtape, printing only file names.
3. List the directory of a disk or DECtape, printing file names, file size, creation date, and protection code.
4. List only certain file(s) or groups of files.
5. Delete a file or group of files.
6. Change the protection of a file.
7. Rename (change the name of) a file.
8. Zero (initialize) a DECtape or RK11 disk directory.
9. Transfer a file or group of files from one device to another.
10. Merge a group of input files into one new output file.
11. List the number of free blocks on device specified.

A detailed description of each feature and its operation appears in either Chapter 3 or Chapter 4.

1.1 CALLING AND EXITING

PIP is called into core, like any other system program under DOS, using the RUN command. The reader should be familiar with certain

DOS commands before attempting to use PIP; in particular, how the RUN command relates to the LOGIN, KILL, and FINISH commands. Briefly, you must be logged in before running PIP, and any previous program must be KILLed.

The RUN command to call PIP into core is typed in response to the Monitor's dot (period) or dollar sign, which denotes DOS Monitor level.

For example:

```
.LOGIN 1,1
$RUN PIP
```

to which PIP will respond with its version number and a number sign (#).

NOTE

All command strings are terminated with the RETURN key.

When PIP prints a number sign, it is ready to receive a command string. After completing the command, PIP will print another number sign, allowing another command string to be typed.

To exit from PIP, type CTRL/C (i.e., hold down the CTRL key while typing the C key). This returns you to the Monitor. At this point you must type the KILL command before you RUN another program. For example, the printout might appear as:

```
.LOGIN 1,1              (log in with your UIC)
$RUN PIP                (call PIP into core)
PIP-11 V00A4A           (PIP identifies itself)
#LP:</DI                (list system device directory on
                        line printer)
                        (directory would appear on the
                        line printer)

#^C                     (CTRL/C to return to Monitor)
.KILL                   (clear core for next program)
$RUN EDIT               (call Edit-11 into core)
    .
    .
    .
```

NOTE

In the examples, we have underlined system program printout to distinguish it from user input which is not underlined.

1.2 RECOVERING FROM ERRORS

When using PIP, the user is likely to receive an error message from time to time.

An A error message indicates a need for operator intervention. A device is out of paper tape or paper, or a similar correctable condition exists. In this case, the user should correct the error condition and give the CONTINUE command. For example:

```
#DT1:ABC<PR:
A00200633200      (high-speed reader is out of tape)
$CONTINUE         (put tape in reader and give CONTINUE
                  command)
#                 (operation has been performed)
```

An F error message indicates a fatal error. Due to the error condition the Monitor cannot continue. The system prints a dollar sign and halts. The user can give a BEGIN or KILL command and continue from that point. For example:

```
#FILE<PR:
F01200267500      (file FILE already exists on system disk)
$BEGIN
#                 (PIP is able to accept another command)
```

When using PIP, there is seldom any need to perform a KILL command and bring a new copy of PIP into core. This procedure is acceptable, but unnecessary. For example:

```
#DT1:ABC<PR:
A00200633200
$↑C
.KILL
$RU PIP
PIP-11 V00400A
#DT1:ABC/DE       (correct error condition)
#DT1:ABC<PR:      (delete file, see note below)
#                 (operation performed)
```

is more time consuming and requires more typing than the two previous examples.

NOTE

In the case above, PIP creates the file ABC on DECtape unit 1, although there is nothing in the file. It is necessary to delete the file before attempting the operation again. This is not necessary if the CONTINUE or BEGIN commands are used.

CHAPTER 2

COMMANDS

All commands in PIP are processed by the DOS Command String Interpreter (CSI). This assures a uniform command language between the various system programs operating under DOS. This section of the manual will describe briefly the command language. The reader is referred to the DOS Monitor Handbook for a complete description of the CSI.

2.1 COMMAND STRINGS

The command string tells PIP which operation or group of operations to perform. For example, the command to transfer a file named ARB.PAL from DECtape unit Ø to the line printer would be:

```
#LP:<DTØ:ARB.PAL
```

Commands are always typed in response to the number sign (#).

Since PIP can perform a large variety of operations, the commands to select the appropriate operation vary. Some operations require two sets of information: one set to describe the output and one set to describe the input. As illustrated in the examples, the input and output are separated from each other by the left angle bracket (<). For example, the following command will list a DECtape directory on the line printer.

```
#LP:<DTØ:/DI
```

There are some operations which need only one specifier; in such a case the specifier acts as output information (except in the case of /DI). For example, to zero (initialize) a new DECtape one need only type:

```
#DTØ:/ZERO
```

It is incorrect to specify the < symbol in such a case. (/ZE is an output switch.)

Commands can consist of a large number of files, limited only by the length of a teleprinter line. To transfer files A, B, C, D,

E, and F from disk to DECtape, one could type:

```
#DTØ:<A,B,C,D,E,F
```

(Notice that disk is the default device.)

To transfer a file ARB.PAL from user [200,200]'s disk area to DECtape unit Ø in formatted ASCII mode, one would type:

```
#DTØ:<ARB.PAL[2ØØ,2ØØ]/FA
```

The general format of a command string is:

```
dev:filnam.ext[uic]/sw1:v1:v2:...vn/sw2:v1:v2:...
```

Each element of a command string is explained below. In the descriptions that follow, the asterisk (*) character will be mentioned. Its usage is fully described in Section 3.1.5. For the interim, the asterisk represents "all". For example,

```
*.PAL
```

means all file names with an extension of PAL.

2.1.1 Device Specification

The device specification, dev:, consists of two letters (listed below), followed by one or more octal digits when required to specify one of multiple units, followed by a colon. The standard device specifications are listed below, where n represents the unit number.

dev	Device
CR:	Card reader
DCn:	Disk, fixed head,(RC11), unit n
DFn:	Disk, fixed head,(RF11), unit n
DKn:	Disk cartridge (RK11), unit n
DTn:	DECtape, unit n
KB:	Keyboard (teleprinter)
LP:	Line printer
MTn:	Magnetic tape, unit n
PP:	High-speed paper tape punch
PR:	High-speed paper tape reader
PT:	Low-speed (teleprinter) paper tape reader/punch

The unit designator (n) is required for multiple devices of the same type, e.g., two DECtapes would be DT1: and DT2:

When a device is not repeated in a command string, the current device is assumed to be the last device specified on the same side (input or output) of the string.

When no device is specified, the system disk is assumed as the default I/O device (except for /DI, /BR, /FR switches where the keyboard is the default output device and the system disk is the default input device). When no unit number is indicated for the device, unit Ø is assumed.

2.1.2 Filename Specification

When required, the filename specification, filnam, consist of from one to six letters or digits, or an asterisk; all characters in excess of six are ignored. The asterisk can be used to represent all file names.

2.1.3 Filename Extension Specification

When required, the filename extension specification, .ext, consists of a period followed by from one to three letters or digits, or an asterisk, which represents all extensions of a named file. A file need not have an extension.

2.1.4 User Identification Code Specification

When required, the user identification code, [uic], consists of a pair of octal numbers separated by a comma and enclosed by square brackets. (The left and right square brackets are typed using SHIFT/K and SHIFT/M respectively.) The left number specifies the user group and the right number specifies the user within the group.

When a UIC does not appear in a command string, the UIC of the user specified with the DOS LOGIN command is assumed.

2.1.5 Switch Specification

When required, the switch specification, /swl:..., consists of a slash followed by one or more letters, and optionally followed by a value specification of octal or decimal digits separated from the switch name by a colon. Switch names can be of any length; however, only the first two characters are ever used.

2.2 SWITCHES

PIP's non-transfer operations are selected by use of switches, and are entered into the command string by preceding the switch with a slash. If more than one switch is used, each is still preceded by a slash.

A summary of PIP's switch options is listed below:

Switch	Name	Operation
/AL	Allocate	Allocate a contiguous file
/BR	Brief Directory	List only filenames and extensions of a directory
/CO	Contiguous	Create a contiguous output file
/DE	Delete	Delete the file
/DI	Directory	List the directory or a subset of the directory (see Chapter 4).
/EN	Enter	Enter the User Identification Code (UIC) in the Master File Directory (MFD)
/FA	Formatted ASCII	Transfer file in formatted ASCII mode
/FB	Formatted Binary	Transfer file in formatted binary mode
/FR	Free	List the number of free blocks left on the device specified
/PR	Protect	Change protection on the file
/RE	Rename	Rename the file
/UN	Unlock	Unlock the User File Directory (UFD) to recover the file
/ZE	Zero	Zero DECtape, magtape, or disk directory

All switches must be typed as the last part of the file specifier, that is, they cannot appear before the filename on which the switch is to operate. Some options require a numeric value to be associated with the switch. For example:

```
/PR
```

is the switch to change the file protection code. The new protection code would follow and be separated from /PR by a colon. For example, a protection change might be written as:

```
#DTØ:ABC/PR:155
```

which would change the protection code of file ABC on DECtape unit Ø from whatever it was before to code 155.

PIP's switch options can be grouped into three categories: preaction switches, action switches, and qualifying switches, as explained below.

2.2.1 Preaction Switches

A preaction switch is permitted in a command string with an action switch. A preaction switch is considered before other switches. Preaction switches are:

ZERO

In the command

#DT:<DT1:ABC/ZE

or

#DT:/ZE<DT1:ABC

the sequence performed in both cases is:

a) DTØ is zeroed (initialized)

b) file ABC is copied from DT1 to DTØ.

2.2.2 Action Switches

Action switches are generally associated with some recognizable operation or action. Only one action switch can be used in any one command string. The action switches are:

ALLOCATE	
BRIEF	PROTECT
DELETE	RENAME
DIRECTORY	UNLOCK
ENTER	FREE

2.2.3 Qualifying Switches

Qualifying switches are used in conjunction with other switches, and are generally used to make a distinction which is otherwise not obvious. Any number of qualifying switches can be used in a single command string. The qualifying switches are:

CONTIGUOUS	FB (Formatted Binary)
	FA (Formatted ASCII)

CHAPTER 3

FILE HANDLING

3.1 FILE TRANSFERS

The transferring of files between devices is one of PIP's primary functions. PIP uses the device independent features of DOS, and in the simplest case transfers files from one peripheral to another. For example, in order to list a file, transfer a copy of the file to the line printer:

```
#LP:<DTØ:FILE1.PAL
```

or

```
#LP:<FILE2.PAL
```

Note that in the second example above, FILE2.PAL is assumed to be on the user's area of the disk, since the disk is the device assumed if none is specified.

File transfers require an output specifier, at least one input specifier, and do not require any switches.

As another example, the following command transfers the data on a tape from the paper tape reader and creates a file named INPUT.TST on the user's area of the disk.

```
#INPUT.TST<PR:
```

The following command duplicates a paper tape.

```
#PP:<PR:
```

There can be only one output file but there can be any number of input files (limited only by the teleprinter line length). For example, to copy three files, ONE.PAL, TWO.PAL, and THREE.PAL, onto DECtape unit Ø, type:

```
#DTØ:<ONE.PAL,TWO.PAL,THREE.PAL
```

3.1.1 Combining Files

When there are several input files involved in a transfer the files

may be transferred without combining, as all previous examples illustrated, or they may be combined into one new output file.

When no output filename is specified with the output device, the input files are copied without combining.

When a filename is specified with the output device, all input files are combined to create the new output file. For example, to combine the three files, SUB1.FTN, SUB2.FTN, and SUB3.FTN, into one new file called SUBS.FTN, you would type:

```
#SUBS.FTN<SUB1.FTN,SUB2.FTN,SUB3.FTN
```

As a more complicated example, consider the files above as existing in a variety of places; then:

```
#DT3:SUBS.FTN<DT1:SUB1.FTN,DF:SUB2.FTN[50,50],DT1:SUB3.FTN[1,1]
```

illustrates the type of syntax one could use.

3.1.2 Modes During Transfers

Files can be created with a variety of modes under DOS. Generally, files are either formatted or unformatted, either ASCII or binary. Formatted generally means that control and checksum information is carried along with the data so the integrity of the file can be assured during usage. Unformatted means that this extra information is not present. In binary mode a full byte of information is always transmitted whereas in ASCII mode generally only seven bits of the byte are passed.

PIP makes no assumptions about the creation mode of any files it transmits. Consequently, the mode assumed for all transfers is unformatted binary unless specified otherwise. This means that formatted files will be transmitted with no benefit of the checksum feature that is available from DOS. It also means, in the case of paper tape, that blank frames, leader and trailer are all transmitted as data. This makes files transferred from the paper tape reader to the disk to be larger than necessary.

3.1.3 Formatted Modes

Switches have been provided to override the assumed unformatted mode. The /FA (Formatted ASCII) and /FB (Formatted Binary) switches will cause the mode of the file associated with the switch to be the one specified. This applies to both input and output files. For example, to transfer the files ABC.DAT and ABC.TST in formatted binary mode, you would type:

```
#DTØ:<ABC.DAT/FB,ABC.TST/FB
```

The purpose of these switches is to notify the user of any read errors. In the case of the paper tape reader, the use of these switches would guard against the case of putting the tape in the reader backwards, as in:

```
#SPRMAN.OBJ<PR:/FB
```

Any attempt to transfer in a mode which the device cannot support is fatal, as in:

```
#LP:/FB<DTØ:ABC
```

which causes a Monitor error message to be printed.

3.1.4 Transfers from Teleprinter

Occasionally, the user may wish to transfer from the teleprinter to another device, as follows:

```
#PP:<KB:
```

The above command is legal, and characters typed are output to the high-speed paper tape punch. To indicate end-of-file with such a command, i.e., to stop transferring to the punch, the DOS convention is to invoke the END command to signal end-of-file. In order to execute the command, the user must be in Monitor mode. Consequently, the sequence is:

```
↑C                          (CTRL/C was typed)
.END                        (RETURN key)
(LINE FEED key)
```

The line feed is necessary in order to force control back to PIP after the END command is executed. PIP returns with a number sign (#) after the above sequence is completed.

The same sequence is necessary when transferring from the low-speed paper tape reader. However, it is necessary to indicate end-of-file from the user's keyboard. For this purpose, the END PT command is given. If the user typed:

```
#DF:FILEIN.MAC< PT:
```

it would be necessary to type:

```
↑C                          (CTRL/C typed)
.END,PT                     (RETURN key)
(LINE FEED key)
```

after the tape has been read and the low-speed reader comes to a stop.

3.1.5 Asterisk in Transfers

The asterisk (*), when used in a command string, provides a convenient means of transferring large numbers of related files. For example, to transfer all load modules from a DECtape to the disk, type:

```
#DF:<DTØ:*.LDA
```

The asterisk is interpreted to mean "all". In the example above, all files with an extension of LDA would be transferred (without combining) onto the disk.

The asterisk can appear in the filename field, the extension field, or both fields. For example, the command:

```
#DTØ:<FORTN.*
```

would transfer all FORTN files, regardless of the extension (including the null extension), to DTØ:. The command:

```
#DF:<DTØ:*.*
```

would copy all files from DTØ: onto the disk.

There is no need to limit the asterisk to one file specifier. The command:

```
#DF:<DTØ:*.PAL,DT1:*.PAL,DT2:*.PAL
```

would transfer all files with extensions of PAL from DTØ:, DT1:, and DT2:.

3.1.6 Contiguous Files

Contiguous files are handled by DOS differently from the regular files, which are called linked files. Blocks of contiguous files are guaranteed by DOS to be physically adjacent, whereas linked files are generally separated by a minimum fixed distance (called the interleave factor) to take advantage of the rotational delay inherent in a disk. Contiguous files are generally used for random accessing of data, a feature which is grossly inefficient with linked files.

Unless specified, file transfers assume linked files and use the corresponding DOS functions to perform the action. The /CO switch is used only to create new contiguous output files. The output field of a PIP command should contain the contiguous specification for a merge operation. The /CO switch is ignored on the input side of the command in a file merge operation (although not in a file copy operation).

When /CO is encountered in the output field of a merge operation, the output file specified is written as a contiguous file. For example:

```
#DK:ABC/CO<DTØ:A,B
```

This command merges the files A and B on DECtape unit Ø to a single contiguous file, ABC, on the disk. (A fatal error occurs if this command is given where the file DK:ABC already exists.)

When transferring files without combining, PIP recognizes the /CO switch associated with any input files and creates the corresponding output file as contiguous. For example, to copy 3 files (currently linked or contiguous) from disk and make them all contiguous on DTØ:, type:

```
#DTØ:<ABC.DAT/CO,DEF.DAT/CO,GHI.DAT/CO
```

Contiguous files are denoted in the directory listing by a C following the number of blocks for the file.

3.2 RENAMING FILES

The RENAME switch is provided to allow the user a means of changing the name of a file. The general form of the command is:

```
dev:FILE.1/RE<dev:FILE.2
```

or

```
dev:FILE.1<dev:FILE.2/RE
```

which changes the name of FILE.2 to FILE.1.

As before, if no device is specified, the disk is assumed. For example:

```
#MAIN.OBJ/RE<TESTX.OBJ
```

changes the name of TESTX.OBJ to MAIN.OBJ.

When renaming, it is mandatory that the same device be used on either side of the < symbol. The following is illegal:

```
#DTØ:FILE1<FILE2/RE
```

because the device for FILE2 was not specified, thus the disk is assumed. The operation would not be performed and an error message would be printed on the user's terminal. The command should be written as:

```
#DTØ:FILE1<DTØ:FILE2/RE
```

3.3 DELETING FILES

The DELETE switch is provided to allow the user a means of deleting one or more files. The simple form of the command is:

```
dev:file/DE
```

For example, the command:

```
#DTØ:MAIN.LDA/DE
```

would delete the file MAIN.LDA from DTØ:.

A number of files can be deleted by specifying a sequence of filenames before the DELETE switch. For example, the command:

```
#MAIN1,DT1:MAIN2.OBJ,DT2:MAIN2.PAL/DE
```

would delete all three files.

The DELETE switch supports the asterisk in the filename or extension fields of file specifications. The asterisk is interpreted as "all", as in *.PAL meaning all files with an extension of PAL. To delete all files with an extension of LDA from a DECtape, type:

```
#DT1:*.LDA/DE
```

To delete all files from the disk, type:

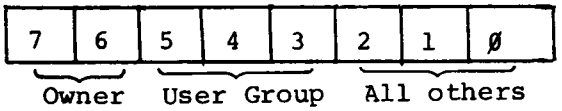
```
#*.*/DE
```

To delete all files from a DECtape, it is recommended that the ZERO switch be used, since that operation is much faster.

3.4 PROTECTING FILES

The PROTECT switch is provided to allow the user a means of changing the protection of a file.

The protection code is specified as an octal value argument to the switch and is treated as three fields corresponding to owner, user group, and all others. Each field is assigned an octal digit as described below.



Owner: Bit 6 = 1 = Owner cannot write on or delete the file. This is a safeguard to prevent inadvertent deletion or over-writing.

Bit 7 = 1 = Protect the file from automatic deletion on FINISH.

User group and all others:

	Function			
Code	Delete	Write	Read	Run
Ø	yes	yes	yes	yes
1		yes	yes	yes
2 or 3			yes	yes
4 or 5				yes
6 or 7				

NOTE

Yes indicates that the operation is allowed. For example, if a file belongs to user [23,10], a protection code of 3 will allow user [12,4] to read or run, but not delete or write on it.

The general form of this command is:

```
dev:fil/PR:value
```

such as

```
#FILE3/PR:355
```

which changes the protection code of FILE3 to 355. The protection switch works for DECtape as well as disk, as in:

```
#DTØ:FILE4/PR:155
```

3.5 RECOVERING FILES

There are a number of infrequent ways in which a file can be left in a state which makes it inaccessible for subsequent processing. For example, if a file is open and a system crash (hardware or software) occurs causing the Monitor to be reloaded, the file will likely be left in an inaccessible state.

Files which are declared inaccessible by DOS will likely have up to three things wrong with them:

1. The LOCK bit in the UFD entry for this file will be set.
2. The USAGE COUNT in the UFD entry for this file will be invalid.
3. Some blocks allocated for this file may not have been marked off in the permanent bit map.

PIP provides a partial solution to this problem with the UNLOCK switch. The function of this switch is to restore the Lock and Usage Count fields so that the file can be read. It does not make an attempt to mark any blocks in the bit map as occupied.

The sequence for recovery is to use the UNLOCK switch, such as:

```
.RUN PIP
#DF:FILE.OLD/UN
```

which allows the file to be accessed.

In order to prevent the blocks from being overwritten, the file should now be transferred to another filename, such as:

```
#DF:FILE.NEW <FILE.OLD
```

which copies the old file and guarantees that it is in the proper state. The old file should then be deleted, such as:

```
#DF:FILE.OLD/DE
```

and the latest file can be renamed, if desired, such as:

```
#DF:FILE.NAM/RE <FILE.NEW
```

A bad link can cause erroneous data to be entered in the link words such that the file may be linked into unknown portions of the disk. Thus, it will be impossible to transfer the file onto a back-up device. Do not attempt to delete the bad file -- it would be disastrous to a portion of the disk. Instead, use Edit-11 and search for the end of the good data, write out the entire last buffer, and close the new output file with the EF command. The bad file should then be renamed to some uncommon name and left as it is until the disk is reinitialized.

3.6 ALLOCATING A CONTIGUOUS FILE

Whenever it is necessary to load a contiguous file from paper tape a blank, contiguous file is first created. The ALLOCATE switch provides a means of creating a blank, contiguous file. This option requires a filename and a decimal integer value (being the number of 64-word (decimal) blocks to be allocated. For example:

```
#DF:FILE/AL:10
```

allocates 640 contiguous words of file storage on the disk. A subsequent PIP command:

```
#FILE <PR:
```

causes the paper tape to be read into the contiguous file previously created.

The ALLOCATE switch can be used to create a contiguous file on disk or DECtape.

CHAPTER 4

DIRECTORY HANDLING

A significant number of PIP features center around the manipulation of files within directories and of the directories themselves.

4.1 ENTERING USER IDENTIFICATION INTO DIRECTORY

The User Identification Code (UIC) must appear in the Master File Directory (MFD) before a user is able to create his own files on the disk or DECtape. This code is generally entered into the MFD by using the ENTER option in PIP, although the system building program creates a few fundamental User File Directories (UFDs).

The UIC of the current user of the system is passed to DOS by the LOGIN command. For example:

```
.LOGIN 1Ø,11
```

sets the current UIC to [1Ø,11].

The UIC of the current user is obtained from DOS and written into a spare slot in the MFD. If no spare slots exist in the MFD, PIP prints a fatal error message, and no attempt is made to extend the MFD. The recommended sequence is then:

```
.LOGIN uic
$RUN PIP
#DF:/EN
```

This procedure is not always necessary for DECtapes because the ZERO switch enters the UIC of the current user when the directory is initialized.

If a user other than the one originally zeroing the DECtape wishes to access files on the tape, he must enter his UIC on the DECtape or explicitly reference the UIC field during each reference.

4.2 DIRECTORY LISTINGS

A directory of all or certain selective files on any file-structured device can be printed on any output device; PIP assumes the teleprinter unless LP: or some other device is specified in the command string. The information listed in a directory is some subset of all the information in the User File Directory (UFD) entry for each file.

The full (/DI) and brief (/BR) directory switches can be used in various ways to determine the type and amount of information to be listed. The asterisk feature (see Section 4.3) can be used with either directory switch for added flexibility.

4.2.1 Full Directories /DI

A full directory contains a title line to identify the device and UIC, current (logged in) date, and all filenames with extensions, number of blocks per file (followed by a C if the file is contiguous), file creation date, and file protection code, plus the total number of blocks and files (for that UIC). For example:

```
#/DI

DIRECTORY DF0: [1,1]

22-JUL-71

MONLIB           508C 00-XXX-70 <377>
LINK11.OVR        36C 21-JUL-71 <233>
PIP    .LDA       70  19-MAR-71 <233>
FCL    .PAL       61  21-JUL-71 <233
PAL11R.OVR        40C 21-JUL-71 <233>
PAL               65  21-JUL-71 <233>
LINK              59  21-JUL-71 <233>

TOTL BLKS:       839
TOTL FILES:        7
```

Notice that the system device (DF0: in this case) was assumed and the user was logged in under UIC 1,1.

Whenever a file is transferred onto disk or DECtape, the date in the directory reflects the date on which the transfer took place, not the date on the original file. The XXX notation in the month field indicates an undated file or an invalid date entry.

The full directories of multiple devices can be obtained

```
#DF:,DT1:/DI

DIRECTORY DF0: [1,1]

24-JUL-71

MONLIB          508C 00-XXX-70 <377>
LINK11.OVR       36C 17-JUL-71 <233>
PIP   .LDA       70  17-JUL-71 <233>
ODT11R.OBJ       51  17-JUL-71 <233>
FORTRN.OVL      152C 24-JUL-71 <233>
FORTRN          118  24-JUL-71 <233>
LINKOB.LDA       96  17-JUL-17 <233>
LINK  .LDA       58  17-JUL-71 <233>
EDIT  .LDA       56  17-JUL-71 <233>
PALOR .LDA      107  17-JUL-71 <233>
PAL   .LDA       65  17-JUL-71 <233>
PAL11R.OVR       40C 17-JUL71  <233>
LIBR  .LDA       41  17-JUL-71 <233>
FORCOM.DGN      112C 17-JUL-71 <00>
EAELIB.LDA      555  19-JUL-71 <233>
EAELIB.OBJ      555  19-JUL-71 <233>

TOTL BLKS:     2620
TOTL FILES:      16


DIRECTORY DT1: [1,1]

24-JUL-71

FORLIB.OBJ      135  01-JUL-71 <233>
EAELIB.OBJ      136  01-JUL-71 <233>
IOL01 .OBJ        1  01-JUL71  <233>
DEFIN .OBJ        1  01-JUL-71 <233>
DVTB06.PAL        7  01-JUL-71 <233>

FREE BLKS:     282
FREE FILES:     51
```

When a full DECtape directory is obtained, the listing is concluded with the total number of free blocks and files, whereas with a disk the number of used blocks and files are reported.

The full directory listing can be limited to produce a partial list of specified files on one or more devices. The following example illustrates the use of the asterisk to list only those files with the extensions specified on the two devices specified.

```
#DF:*.OVR,*.OBJ,DT1:*.PAL/DI


DF0:

LINK11.OVR        36C 17-JUL-71 <233>
PAL11R.OVR        40C 17-JUL-71 <233>

ODT11R.OBJ         51  17-JUL-71 <233>
FAELIR.OBJ        555  19-JUL-71 <233>

DT1:

DVTR06.PAL          7  01-JUL-71 <233>
```

Notice that the UIC, date, and number of blocks and files for that UIC are not reported in partial /DI listings.

Full information about a particular file can be obtained by specifying the filename and extension with the /DI switch. For example:

```
#FTN000.OVL/DI


DK0:

FTN000.OVL        10C 15-JUL-71 <233>
```

This obtains all information about FTN000.OVL if it is in the DK0: directory. (If the file is not in the directory, there is no listing, but no error message. Likewise, FTN000.OVL/BR allows the user to verify the existence of the file.)

Any UIC directory on any device can be listed by specifying that UIC in the command to PIP. For example, if a user is logged in under UIC 200,200, he can give the following command (the results are shown below):

```
#DT1:[1,1]/DI


DIRECTORY DT1:  [1,1]

21-JUL-71

FORCOM.DGN        27C 01-JUL-71 <233>
FORTRN.08K        23  01-JUL-71 <233>
OVL08K.LDA        27  01-JUL-71 <233>
OVL18K.LDA        26  01-JUL-71 <233>
OVL28K.LDA        23  01-JUL-71 <233>
OVL38K.LDA        26  01-JUL-71 <233>
OVL48K.LDA        28  01-JUL-71 <233>
OVL58K.LDA        28  01-JUL-71 <233>
OVL68K.LDA        25  01-JUL-71 <233>
FORTRN.12K        23  01-JUL-71 <233>
OVL0  .LDA        27  01-JUL-71 <233>
OVL1  .LDA        26  01-JUL-71 <233>
OVL2  .LDA        23  01-JUL-71 <233>
OVL3  .LDA        26  01-JUL-71 <233>
OVL4  .LDA        28  01-JUL-71 <233>
OVL5  .LDA        28  01-JUL-71 <233>
OVL6  .LDA        25  01-JUL-71 <233>

FREE BLKS:      123
FREE FILES:      39
```

If the user requests information on any particular file or uses the asterisk feature where no such file exists, no error message is given. The directory is considered blank; and a blank line is printed. In the example below there is no *.OVR file on DTØ.

```
#DF:*.OVR,DTØ:*.OVR/DI


DFØ:

LINK11.OVR        36C 21-JUL-71 <233>
PAL11R.OVR        40C 21-JUL-71 <233>

DTØ:
```

4.2.2 Brief Directories, /BR

The brief directory switch can be used to list only the device name, filenames, and extensions. The asterisk and specific filenames can be used with the brief directory switch to obtain partial brief directories, as in section 4.2.1. The following PIP example demonstrates the correspondence between /DI and /BR directory listings.

```
#*.OVL/DI

DK0:

FTN000.OVL        100  15-JUL-71  <233>
FTN001.OVL         9C  15-JUL-71  <233>
FTN002.OVL         7C  15-JUL-71  <233>
FTN003.OVL         9C  15-JUL-71  <233>
FTN004.OVL        10C  15-JUL-71  <233>
FTN005.OVL        11C  15-JUL-71  <233>
FTN006.OVL         8C  15-JUL-71  <233>


#*.OVL/BR

DK0:

FTN000.OVL
FTN001.OVL
FTN002.OVL
FTN003.OVL
FTN004.OVL
FTN005.OVL
FTN006.OVL
```

An example of the /BR switch with the system device assumed is shown below:

```
#/BR

DF0:

MONLIB
LINK11.OVR
PIP    .LDA
FCL    .PAL
PAL11R.OVR
PAL
LINK
```

Other examples of /BR switch usage are shown below:

```
#FOR.E/BR

DF0:

FOR    .E
```

This is the most efficient way to verify the existence of a particular file on a given device.

```
#DT0:,DT1:/BR


DT0:

FORCOM.DGN
FORTRN.08K
OVL08K.LDA
OVL18K.LDA
OVL28K.LDA
OVL38K.LDA
OVL48K.LDA
OVL58K.LDA
OVL68K.LDA
FORTRN.12K
OVL0   .LDA
OVL1   .LDA
OVL2   .LDA
OVL3   .LDA
OVL4   .LDA
OVL5   .LDA
OVL6   .LDA

DT1:

FORLIB.OBJ
EAFLIB.OBJ
IOL01  .OBJ
DEFIN  .OBJ
DVTR06.PAL
```

4.2.3 Free Blocks, /FR

Since a full directory listing of a system disk provides the user only with a summary of the number of blocks used by his current UIC, the /FR switch is available for determining the number of free blocks remaining on the entire device. For example:

```
#/FR


DF0:
FREE BLKS:  5513
```

This switch can be used with any device, but is generally only necessary with disk, as the information is not provided automatically with a directory listing. Further examples of the /FR switch follow:

```
#/FR


DKØ:
FREE BLKS:  4113




#DF:,DTØ:/FR


DFØ:
FREE BLKS:  3243

DTØ:
FREE BLKS:   282
```

4.3 ASTERISK IN DIRECTORY LISTINGS

The asterisk (*) feature for filenames and extensions works for directory listings, both normal and brief form. For example, the command

```
#*.PAL/BR
```

gives a brief directory listing for all files on the user's disk area with an extension of PAL. The asterisk can appear in the filename field, the extension field, or both. In addition, groups of file specifiers can be arranged to provide sequential listings, as in

```
#DTØ:*.MAC,DF:FORTN.*/DI
```

4.4 DECTAPE AND RK11 DISK INITIALIZATION

4.4.1 DECtape

In order to initialize a new DECtape with the basic file structure information required by the DOS Monitor, the ZERO switch is provided.

The function creates the permanent bit maps, the file bit maps, the MFD, and a UFD for the user currently running this program.

This switch may be used either by itself, as in:

```
#DTØ:/ZE
```

or in combination with other switches, as in:

```
#DTØ:ONE/ZE<DT1:A,B,C
```

which zeroes DTØ: first, then creates file ONE on DTØ: by concatenating files A, B, and C from DT1:. When used in combination with other switches, ZERO is performed before any other implied actions.

4.4.2 RK11 Disk

The Zero switch can be used to initialize the RK11 disk. As with DECtape, the following four actions occur during initialization, in the order shown below:

1. The entire disk cartridge is zeroed;
2. The first and second MFD blocks are written;
3. The UIC of the current user (and only this UIC) is entered in the MFD; then
4. Bit map blocks are written.

The RK11 disk is initialized by typing:

```
#DKn:/ZE
```

where n is the octal number of the appropriate cartridge (or omitted if DKØ is meant). PIP responds by printing:

```
CONFIRM:
```

to which type either H (for high-density) or L (for low-density) followed by the RETURN key. Should the user decide not to initialize the disk, he types the SPACE bar before typing the RETURN key.

High-density cartridges are marked 2200 BPI, and low-density cartridges are marked 1100 BPI.

CHAPTER 5

ERROR MESSAGES

Error messages conform to the standard for error handling as defined in the DOS Monitor handbook. Errors are of the form:

Sxxx

where the S indicates a system program error.

Error messages unique to PIP are listed below. Consult the DOS Monitor handbook for a complete description of all error messages.

Error Message	Reason
S203	Too many switches or an illegal switch
S204	Too many output specifiers
S205	Too many output files
S231	Illegal command; file-structured device required
S232	More than one action switch specified
S233	UIC not in MFD
S234	No file name where one is required
S235	No files exist in UFD
S237	File not found
S240	Insufficient space for file
S241	MFD is full
S242	Meaningless command; no action taken
S252	Filename given where none allowed
S254	User tried to zero the system resident disk
I354	DK11 disk cartridge was not zeroed (illegal response to CONFORM:)

APPENDIX A

SUMMARIES

Switch	Name	Example	Operation
/AL	Allocate	#DF:FILE/AL:1Ø	640 (10 x 64) contiguous words of file storage is allocated on disk.
/BR	Brief Directory	#DTØ:/BR	List all filenames and extensions of directory for DECtape Ø onto the user Teletype.
/CO	Contiguous	#DTØ:<ABC/CO	Transfer file ABC as contiguous file from disk to DECtape Ø.
		#DTØ:AD/CO<ABC,DEF	Merge files ABC and DEF on disk as one contiguous file on DECtape Ø.
/DE	Delete	#MAIN/DE	Delete file MAIN from disk.
		#DTØ:ABC,DT1:DEF/DE	Delete files ABC and DEF from DECtapes Ø and 1 respectively.
/DI	Directory	#DTØ:/DI	List all filenames, extensions, size, creation date, and protection codes of directory for DECtape Ø onto the user's Teletype.
/EN	Enter	#DF:/EN	Enter current user's UIC in MFD on disk.
/FA	Formatted ASCII	#LP:/FA<DTØ:ABC	Copy file ABC from DECtape Ø to line printer in formatted ASCII mode.
/FB	Formatted Binary	#DTØ:<ABC/FB	Copy file ABC from disk to DECtape Ø in formatted binary mode.
/FR	Free	#DT1:/FR	List the number of free blocks remaining on DECtape unit 1.
/PR	Protect	#ABC/PR:355	Change protection code of disk file ABC to 355.
/RE	Rename	#ABC/RE<DEF	Rename disk file DEF to ABC.
		#DTØ:GHI/RE<DTØ:JKL	Rename DECtape Ø file JKL to GHI.
/UN	Unlock	#DF:FILE.OLD/UN	Unlock FILE.OLD after a system failure so that FILE.OLD can be accessed.
/ZE	Zero	#DTØ:/ZE	Erase all files from directory of DECtape Ø.

Device Specifications

dev	Device
CR:	Card reader
DCn:	Disk, fixed head, unit n
DFn:	Disk, fixed head, unit n
DKn:	Disk cartridge (RK11), unit n
DTn:	DECtape, unit n
KB:	Keyboard (teleprinter)
LP:	Line printer
MTn:	Magnetic tape unit n
PP:	High-speed paper tape punch
PR:	High-speed paper tape reader
PT:	Low-speed (teleprinter) paper tape reader/punch

APPENDIX B

HELPFUL HINTS

There are a few things which are obvious to experienced DOS users, but the beginner often has to learn them empirically. This section is meant to help the beginner avoid some common pitfalls.

1. If you are having difficulty running PIP, make sure that:

   a. you are logged in,

   b. you did a KILL command after running a previous program,

   c. the load module (PIP.LDA) is on the disk, or,

   d. you are accessing the proper UIC field on the DECtape, if attempting to run from DECtape, i.e., you should be logged in under the same UIC as shown on the directory of the DECtape.

2. If you are having difficulty accessing a DECtape file which you are positive is there, you are probably logged in under a different UIC than appears on the DECtape. In this case, either:

   a. Explicitly specify the UIC in the command,
   b. FINISH and LOGIN under the proper number, or,
   c. ENTER your UIC on the DECtape (see Section 4.1).

3. If you are having difficulty transferring to the line printer, make sure the printer is on-line; set the switch to ON LINE.

4. If you are having difficulty reading a file and there has recently been a software or hardware crash, read Section 3.6, Recovering Files.

5. If you continue to get file structure errors on output files, you are probably trying to create a file which already exists. DOS does not allow this, so delete the file first.

6. If you are logged in for the very first time and cannot create any files, it is because you have no User File Directory (UFD) in the Master File Directory (MFD). To create one, Log in, call PIP and type:

```
#DF:/EN
```

and you are all set.

7. If you cannot write anything on a DECtape even though the write switch is on, it is likely that:

   a. the DECtape belongs to someone else and he does not want you to write on it accidentally, or,

   b. the tape has not been certified.

   If the DECtape does belong to someone else, you may still write on it by using the ENTER switch to put your directory on the DECtape, as in:

```
#DTØ:/EN
```

   Uncertified DECtapes can be certified using a diagnostic program (TC11 DECtape Formatter MAINDEC-11-YPTB-PB) supplied with the maintenance programs.

APPENDIX C

MAGTAPE OPERATIONS

This appendix describes those changes made to VØØ4A PIP to incorporate several industry compatible magtape operations. The version of PIP described is distributed as VØØ5A PIP.

C.1 GENERAL MAGTAPE INFORMATION

Industry compatible magnetic tape is supported by PIP for the following operations:

a. Directory listings

```
#MT1:/DI
#MTØ:/BR
```

b. File transfer operations

```
#MT1:FILE<DT2:ABC
#MT:MERGE<ABC,DTØ:FACT,MT1:DATA
```

c. Directory initialization

```
#MT1:/ZE
```

d. Rewind, and Rewind and Unload (see sections C.2 and C.3).

Transfer and directory operations include all asterisk features used for other file-structured devices. In addition, on magtape it is possible to say:

```
#MT1:[*,*]/DI
```

or

```
#MT1:[*,*]/BR
```

This allows the user to obtain a complete listing of all files on the specified magtape(s). When this option is used, the UIC specification is printed as part of the file information. For example:

```
#MTØ:[*,*],MT2:[*,*]/DI

DIRECTORY MTØ:  [*,*]

13-SEP-71

DDT    .PAL     59  11-SEP-71 <233>   [2ØØ,2ØØ]
EDIT            14  13-SEP-71 <233>   [1,1]

TOTL BLKS:      73
TOTL FILES:      2

DIRECTORY MT2:  [*,*]

13-SEP-71

ODT    .OBJ     13  13-SEP-71 <233>   [1,1]

TOTL BLKS:      13
TOTL FILES:      1
```

C.2 REWIND

The Rewind switch is applicable only to magtape units. This switch causes the tape to be wound back on its original reel. Following this option the magtape unit(s) can still be addressed. For example:

```
#MT1:/RW
```

or

```
#MT1:,MT2:/RW
```

are acceptable commands.

C.3 REWIND and UNLOAD

The Rewind and Unload switch is applicable only to magtape units. This switch causes the tape to be wound back on its original reel and the magtape unit to be switched off-line. Following this option the MAG-tape unit(s) cannot be addressed for further processing. For example:

```
#MT:/RU
```

or

```
#MT2:,MT3:/RU
```

C.4 ZERO

It is mandatory that a magtape be zeroed before use. The /ZE switch writes two end-of-file marks on the tape so that it can be handled correctly by DOS.

APPENDIX D

PROGRAMMING CAUTIONS

D.1 ASTERISK FEATURE

Versions V004A and V005A of PIP do not allow the use of the asterisk in renaming operations or in the output side of a transfer command operation. Attempts to use the asterisk under these conditions result in a file called space-A-B (i.e., "␣AB") in the field corresponding to the asterisk in the command. Space-A-B files cannot be deleted with the /DE switch. For example:

Legal form	Results	Illegal Form	Results
#DF:<DT1:*.*	Moves all files to DF:	#DF:*.*<DT1:*.*	Creates file ␣AB.␣AB
		#DF:*.OBJ<DT1:*.*	Creates file ␣AB.OBJ
		#DF:*.PAL<*.P11/RE	Creates file ␣AB.PAL
		#DF:*.*<*.TST/RE	Creates file ␣AB.␣AB

In order to remove the space-A-B file, rename the file using the illegal form as the current name, and then either use the renamed file or delete it. For example:

```
                                                    Comments
#*.MB/RE<*.LDA                                  This is an illegal operation.
#/DI
DIRECTORY DKØ: [2ØØ,2ØØ]
28-OCT-71
 AB    .MB       26 28-OCT-71 <233>          The file ␣AB.MB was created.
TOTL BLKS:      26
TOTL FILES:      1
#A/RE<*.MB                                      Change the name to a legal
#/DI                                            form.
DIRECTORY DKØ: [2ØØ,2ØØ]
28-OCT-71
A              26 28-OCT-71 <233>
TOTL BLKS:     26
TOTL FILES:     1
```

D.2 NUMBER OF UIC'S ON RF11 DISK

There is currently space for 15 different UIC's on the RF11 disk system.

D.3 TRANSFER OF ASCII CARDS

To transfer ASCII cards from the card reader to disk or DECtape, the /FA switch must be used.

D.4 ALLOCATE SWITCH

The ALLOCATE switch allows the user to create an empty contiguous file of specified length. In this version of PIP it is not possible to transfer an existing file into this empty, contiguous file.

Accessing files, 4-1
Action switches, 2-5
ALLOCATE switch, 3-7
Allocating contiguous files, 3-7
ASCII mode, 3-2
Asterisk (*) usage, 2-2, 3-4, 3-6, D-1

Binary mode, 3-2
Blank frames, 3-2
BRIEF switch, 4-6

Calling PIP, 1-1
Changing file name, 3-5
/CO (contiguous) switch, 3-5
Combining files, 3-1
Command
  language, 2-1
  String Interpreter (CSI), 2-1
  strings, 2-1
Commands
  END, 3-3
  FINISH, 1-1
  KILL, 1-1, 1-2, B-1
  LOGIN, 1-1, 4-1
  RUN, 1-1
Contiguous files, 3-4
  allocation, 3-7
CTRL/C, 1-2

DECtape
  directory sample, 4-2
  initializing or zeroing, 2-1,4-4
DELETE switch, 3-6
Deleting files, 3-6
Device specification, 2-2, A-2
Directories, sample of disk and DECtape, 4-2
Directory listings, 4-1
  subset, 4-3
Disk directory sample, 4-2
Dollar sign ($) usage, 1-1
Dot or period (.) usage, 1-1
Duplicating paper tape, 3-1

END command, 3-3
End-of-file, 3-3
Entering User Identification into directory, 4-1
Error messages, 4-1, 5-1
  monitor, 3-3
  see also Helpful Hints, B-1
Exiting from PIP, 1-2
Extension PAL, 2-2

/FA (Formatted ASCII) switch, 3-3
/FB (Formatted Binary) switch, 3-3
Features of PIP, 1-1
File
  handling, 3-1
  protection, 3-7
  recovery, 3-8
  transfers, 3-1
Filename
  extension, 2-3
  specification, 2-3
FINISH command, 1-1
Formatted
  files, 3-2
  modes, 3-3

Initializing
  DECtape, 2-1,4-9
  file directory, 4-1
Interleave factor, 3-4

KILL command, 1-1, 1-2, B-1

Leader tape, 3-2
Left angle bracket (<) usage, 2-1, 4-2
Linked files, 3-4
LOCK bit, 3-8
LOGIN command, 1-1, 4-1

Master File Directory (MFD), 4-1
Modes during transfers, 3-2

Number sign (#), 1-2, 2-1

Overwriting, 3-8

PAL extension, 2-2
Period or dot (.) usage, 1-1
Permanent bit map, 3-8
Protect switch, 3-7
Protecting files, 3-7
Protection code, 3-7
Protection switch, 3-8
/PR (change file protection code) switch, 2-4

Qualifying switches, 2-5

Random accessing, 3-4
Recovering files, 3-8
RENAME switch, 3-5
Renaming files, 3-5
RETURN key, 1-2
RUN command, 1-1

Sample directories, disk and DECtape, 4-2, 4-3
Slash (/) usage, 2-3
Square bracket usage, 2-3
Summaries
  device specifications, A-2
  switch, A-1
  switch options, 2-4
Switch names, 2-3
Switches, 2-3, 3-3
  action, 2-5
  ALLOCATE, 3-7
  BRIEF, 4-3
  /CO, 3-5
  DELETE, 3-6
  /FA, 3-3
  /FB, 3-3
  options summary, 2-4
  /PR, 2-4
  PROTECT, 3-7
  qualifying, 2-5
  RENAME, 3-5
  summary, A-1
  UNLOCK, 3-8
  ZERO, 3-6, 4-1

Trailer tape, 3-2
Transfer modes, 3-2
Transferring
  files, 2-1, 2-2
  files to line printer, 3-1
  files from low speed paper tape reader, 3-3
  files from paper tape reader, 3-1
  related files, 3-4
  several input files, 3-1
  files from teleprinter, 3-3

UNLOCK switch, 3-8
USAGE COUNT, 3-8
Unformatted files, 3-2
User Identification Code (UIC), 2-3, 4-1
User identification entry, 4-1, 4-2

ZERO switch, 3-6, 4-1, 4-4
Zeroing DECtape, 2-1, 4-9
Zeroing Disk, 4-10

DEC-11-ZLDC-D

PDP-11

LINK-11 LINKER

AND

LIBR-11 LIBRARIAN

PROGRAMMER'S MANUAL

for the

Disk Operating System

SOFTWARE SUPPORT CATEGORY

The software described in this document is supported by DEC under Category I, as defined on page iv of this document.

For additional copies, order No. DEC-11-ZLDC from Digital Equipment Corporation, Direct Mail, Bldg. 1-1, Maynard, Massachusetts 01754

First Printing, May 1971
Second Edition, September 1971
Third Edition, February 1972

Your attention is invited to the last two pages of this document. The "How to Obtain Software Information" page tells you how to keep up-to-date with DEC's software. The "Reader's Comments" page, when filled and mailed, is beneficial to both you and DEC; all comments received are acknowledged and are considered when documenting subsequent manuals.

Copyright © 1971, 1972 by Digital Equipment Corporation

This document is for information purposes and is subject to change without notice.

Associated Documents:

PDP-11 Disk Operating System Monitor,
Programmer's Handbook, DEC-11-MWDC-D

PDP-11 FORTRAN IV,
Programmer's Manual, DEC-11-KFDA-D

PDP-11 PAL-11R Assembler,
Programmer's Manual, DEC-11-ASDC-D

PDP-11 Edit-11 Text Editor,
Programmer's Manual, DEC-11-EEDA-D

PDP-11 ODT-11R Debugging Program,
Programmer's Manual, DEC-11-OODA-D

PDP-11 PIP File Utility Package,
Programmer's Manual, DEC-11-PIDB-D

Trademarks of Digital Equipment Corporation include:

DEC	PDP-11
DECtape	RSTS-11
DIGITAL (logo)	RSX-11
COMTEX-11	UNIBUS

PREFACE

This manual describes the features and operation of the Link-11 Linker and the Libr-11 Librarian; both are system programs for the PDP-11 Disk Operating System (DOS). The reader is expected to be familiar with the DOS Monitor, DOS Assembler, and PIP programs, as described in their respective documents listed on the preceding page.

The manual has two major parts:

Part I	describes the Link-11 Linker
Part II	describes the Libr-11 Librarian

A Master Table of Contents follows this page, and each major part begins with a separate detailed Table of Contents. Chapters are numbered sequentially throughout the manual, and for quick reference purposes the manual is concluded with a comprehensive index.

In addition to the Link-11 Linker and the Libr-11 Librarian, the PDP-11 Disk Operating System software includes:

DOS Monitor	Edit-11 Text Editor
FORTRAN IV	ODT-11R Debugging Program
PAL-11R Assembler	PIP-11 File Utility Package

DOCUMENTATION CONVENTIONS USED HEREIN

1. System program printout is underlined to differentiate it from user-typed commands, which are not underlined. However, large sections of obvious system program printout (listings, maps, etc.) are not underlined.

2. All commands are terminated with the nonprinting RETURN key, causing a carriage return/line feed operation. Where necessary, <CR> is used to represent the RETURN key.

3. In certain examples, sysdev: is used to represent the system device name of the particular installation's system disk, i.e., DK:, DF:, and DC:.

4. The CTRL/key combinations are typed by holding down the CTRL key while typing the required letter key. For examples, CTRL/C ensures that the Monitor will accept the next command (but will not necessarily immediately stop a job in progress), and CTRL/U deletes the line on which it is typed.

SOFTWARE SUPPORT CATEGORIES

Digital Equipment Corporation (DEC) makes available four categories of software. These categories reflect the types of support a customer may expect from DEC for a specified software product. DEC reserves the right to change the category of a software product at any time. The four categories are as follows:

CATEGORY I
Software Products Supported at no Charge

This classification includes current versions of monitors, programming languages, and support programs provided by DEC. DEC will provide installation (when applicable), advisory, and remedial support at no charge. These services are limited to original purchasers of DEC computer systems who have the requisite DEC equipment and software products.

At the option of DEC, a software product may be recategorized from Category I to Category II for a particular customer if the software product has been modified by the customer or a third party.

CATEGORY II
Software Products that Receive Support for a Fee

This category includes prior versions of Category I programs and all other programs available from DEC for which support is given. Programming assistance (additional support), as available, will be provided on these DEC programs and non-DEC programs when used in conjunction with these DEC programs and equipment supplied by DEC.

CATEGORY III
Pre-Release Software

DEC may elect to release certain software products to customers in order to facilitate final testing and/or customer familiarization. In this event, DEC will limit the use of such pre-release software to internal, non-competitive applications. Category III software is only supported by DEC where this support is consistent with evaluation of the software product. While DEC will be grateful for the reporting of any criticism and suggestions pertaining to a pre-release, there exists no commitment to respond to these reports.

CATEGORY IV
Non-Supported Software

This category includes all programs for which no support is given.

MASTER TABLE OF CONTENTS

PART I	LINK-11 LINKER	
Chapter 1	Introduction	1-1
Chapter 2	Input and Output	2-1
Chapter 3	Operating Procedures	3-1
Chapter 4	Error Handling and Messages	4-1
Chapter 5	Summary of Link-11 Switches	5-1
Chapter 6	Program Overlay Facility	6-1
Chapter 7	LINK-11 Under RSX-11C	7-1
PART II	LIBR-11 LIBRARIAN	
Chapter 8	Introduction	8-1
Chapter 9	Operating Procedures	9-1

APPENDIX A

Sample uses of Linker for FORTRAN, assembly language programs and overlay systems.

INDEX

PART I

LINK-11 LINKER

Chapter 1.		Introduction to Link-11 Linker	1-1
	1.1	Absolute and Relocatable Program Sections	1-2
	1.2	Global Symbols	1-3
	1.3	Relinking Link-11	1-3
Chapter 2		Input and Output	2-1
	2.1	Input Module	2-1
	2.2	Output Module	2-1
	2.2.1	Absolute Loader	2-1
	2.2.2	DOS Monitor Loader	2-2
	2.3	Load Map	2-2
	2.4	Core Library Symbol Table File	2-3
Chapter 3		Operating Procedures	3-1
	3.1	Loading	3-1
	3.2	Command String	3-1
	3.2.1	Switches	3-2
	3.2.1.1	Top and Bottom Switches	3-2
	3.2.1.2	Undefined Globals Switch	3-3
	3.2.1.3	Tapes Switch	3-3
	3.2.1.4	Concatenate Switch	3-4
	3.2.1.5	ODT Switch	3-4
	3.2.1.6	Transfer Address Switch	3-4
	3.2.1.7	Exit Switch	3-5
	3.2.1.8	Library Switch	3-5
	3.2.1.9	General Notes on Switches	3-5
	3.3	Library Searches	3-6
	3.3.1	User Libraries	3-6
	3.3.2	Monitor Library	3-6
	3.4	Programming Notes and Cautions	3-7
	3.4.1	Programming Notes	3-7
	3.4.2	Cautions	3-10
Chapter 4		Error Handling and Messages	4-1
	4.1	Restarting	4-1
	4.2	Warning Error Messages	4-1
	4.3	Action Request Messages	4-3
	4.4	Fatal Linking Error Messages	4-3

CONTENTS

Chapter 5		Summary of Link-11 Switches	5-1
Chapter 6		Program Overlay Facility	6-1
	6.1	Purpose of an Overlay Facility	6-1
	6.2	Communication Among Resident and Overlay Routines	6-2
	6.3	Format Conversions	6-2
	6.4	Link, the Run-Time Overlay Supervisor	6-4
	6.5	Calling an Overlay File	6-5
	6.5.1	Overlay Transfer Paths	6-7
	6.5.2	Search for Overlay Files	6-7
	6.6	Operating Procedures	6-8
	6.6.1	Creating the Core Resident Module	6-8
	6.6.2	Creating the Overlay Files	6-8
	6.7	Error Procedures and Messages	6-9
	6.8	Assembly Language Overlays	6-11
	6.8.1	Global Declarations	6-11
	6.8.2	Stack Usage	6-11
	6.8.3	Register Usage	6-12
	6.8.4	COMMON Communication	6-12
	6.9	Programming Notes	6-13
	6.10	A Summary of the Major Features of the Overlay Facility	6-14
	6.11	Programming Example of Overlay System Creation	6-15
Chapter 7		Link-11 Under RSX-11C	7-1
	7.1	Task Limits	7-1
	7.2	Device Names	7-2
	7.3	Relinking the Linker	7-2
	7.4	Real-Time Header Generation	7-3
	7.5	Core Library Symbol Table	7-4
	7.5.1	Core Library Linker Procedures	7-5

CHAPTER 1

INTRODUCTION TO LINK-11 LINKER

The PDP-11 Disk Operating System software includes the Link-11 Linker; a system program for linking and relocating user programs assembled by the DOS Assembler. Link-11 enables the user to separately assemble his main program and various subprograms without assigning an absolute address for each segment at assembly time.

The binary output (object module) of each assembly can be processed by Link-11 to:

- Relocate each object module and assign absolute addresses.
- Link the modules by correlating global symbols defined in one module and referenced in another module.
- Produce a load map which displays the assigned absolute addresses.
- Create a load module which can subsequently be loaded (by the Monitor or the Absolute Loader) and executed.

The advantages of using Link-11 include:

- The source program can be divided into segments (usually subroutines) and assembled separately. If an error is discovered in one segment, only that segment needs to be reassembled. Link-11 can then link the newly reassembled object module with other object modules.
- Absolute addresses need not be assigned at assembly time; the Linker automatically assigns absolute addresses. This keeps programs from arbitrarily overlaying each other, and also allows subroutines to change size without influencing the placement of other routines but not affecting their operation.
- Separate assemblies allow the total number of symbols to exceed the number allowed in a single assembly.
- Internal symbols (which are not global) need not be unique among object modules. Thus, naming rules are required for global symbols only, when different programmers prepare separate subroutines for a single run time system.

- Subroutines may be provided for general use in object module form to be linked into the user's program.

Link-11 requires at least an 8K PDP-11 with a disk and a teleprinter. DECtape, high-speed paper tape reader and punch, a line printer and more core may be used if available. The DECtape and the high-speed reader/punch significantly speed up the linking process. A line printer provides a fast display device for the load map listing.

The instructions and assembler directives of the DOS Assembler are used herein to describe the operation of Link-11. However, Link-11 can link and relocate any object modules which conform to its linking format (consult your nearest DEC representative for the linking format).

1.1 ABSOLUTE AND RELOCATABLE PROGRAM SECTIONS

A program assembled by the DOS Assembler can consist of an absolute program section, declared by the .ASECT assembler directive, and relocatable program sections declared by the .CSECT assembler directive. The Assembler assumes a .CSECT directive at the beginning of the source program. The instructions and data in relocatable sections are assigned normally at the high end of memory. The assignment of addresses can be influenced by command string options (see Section 3.2). The Linker appropriately modifies all instructions and/or data as necessary to account for the relocation of the control sections (as declared by .CSECT's).

Link-11 handles the absolute section as well as the named and unnamed control sections. The unnamed control section is internal to each object module. That is, every object module can have an unnamed control section but the Linker treats them independently. Each is assigned an absolute address such that they occupy mutually exclusive areas of memory. Named control sections, on the other hand, are treated globally. That is, if different object modules each have control sections with the same name, they are all assigned the same absolute load address and the size of the area reserved for loading of the section is the size of the largest. Thus, named control sections allow for the sharing of data and/or instructions among object modules. This is similar to the handling and function of COMMON in FORTRAN IV. The names assigned to control sections are global and can be referenced as any other global symbol.

1.2 GLOBAL SYMBOLS

Global symbols provide the links, or communication, between object modules. With the DOS Assembler these symbols would be created with the .GLOBL assembler directive. Symbols which are not global are called internal symbols. If the global symbol is defined (as a label or by direct assignment) in an object module, it is called an entry symbol and other object modules can reference it. If the global symbol is not defined in the object module, it is an external symbol and is assumed to be defined (as an entry symbol) in some other object module.

As the Linker reads the object modules it keeps track of all global symbol definitions and references. It then modifies the instructions and/or data which reference the global symbols.

1.3 RELINKING LINK-11

Link-11 is provided as a system program with the PDP-11 Disk Operating System.

It is available from DEC in the following formats:

Two absolute load modules for 8K systems.
Two object modules for relinking (see Section 3.5).
Seven ASCII source tapes.

These enable you to relink Link-11 using the supplied Link-11 load modules to load into any 8K or larger system. The resulting Linker will assume a top of memory corresponding to the system configuration; this can be over-ridden using the T (top) or B (bottom) switches (see Section 3.2.1.1).

The top address assumed by the Linker is:

nnn460

where nnn is explained below.

nnn	for Memory Size
37	8K
57	12K
77	16K
117	20K
137	24K
157	28K

CHAPTER 2

INPUT AND OUTPUT

2.1 INPUT MODULE

Link-11's input is the object module. This is the output of the DOS Assembler. The Linker reads each object module twice, thus, it is a two-pass processor.

On pass 1, the Linker reads each object module to gather enough information so that absolute addresses can be assigned to all relocatable sections and all globals can be assigned absolute values. This information appears in the global symbol directory (GSD) of the object module.

On pass 2, the Linker reads all of each object module and produces the load module. The data gathered on pass 1 guides the relocation and linking process on pass 2.

2.2 OUTPUT MODULE

The normal output of the Linker is a load module which can be loaded and run. A load module consists of formatted binary blocks of absolute load addresses and object data as specified for the Absolute Loader and the DOS Monitor Loader. The first few words of data will be the communications directory (COMD), and will have an absolute load address equal to the lowest relocated address of the program.

2.2.1 Absolute Loader

The Absolute Loader will load the COMD at the specified address, but then the program will overlay the COMD. The overlaying of the COMD by the relocated program allows the Absolute Loader to handle load modules with a COMD. However, a problem arises if a load module is to be loaded by the Absolute Loader and either of the following conditions exist:

1. The object modules used to construct the load module contain no relocatable code, or
2. The total size of the relocatable code is less than the size of the COMD.

In either case, there would not be enough relocatable code to overlay the COMD, which means that the COMD will load into parts of core not intended to be altered by the user. The Linker will select the COMD's load address such that the COMD will be against the current top (see T switch in Section 3.2). If the top is very low, the Linker will not allow the COMD to be loaded below address Ø; it will load it up from Ø.

2.2.2 DOS Monitor Loader

The DOS Monitor Loader will load the COMD where the Monitor wants it. The end of the load module will be indicated by a TRA (transfer address) block; that is, a block containing only a load address. The byte count in the formatted binary block will be 6; on all other blocks the byte count will be greater than 6. If the TRA is not specified by a switch, it is assumed by the Linker to be the first even transfer address encountered. Thus, if four object modules are linked together and if the first and second have a .END statement without an address, the third a .END A, and the fourth a .END B, the transfer address would be A of module three.

2.3 LOAD MAP

The load map produced by Link-11 provides several types of information concerning the load module's make-up. The map begins with the transfer address and the low and high limits of the relocatable code. Then there is a section of the map for each object module included in the linking process. Each of these sections begins with the module's name followed by a list of the control sections and their entry points. For each control section, the base of the section (its low address) and its size (in bytes) is printed to the right of the section name (enclosed in angle brackets). Following each section name is an alphabetically ordered list of entry points and their addresses. A list of any undefined symbols for each object module concludes the load map.

Note that modules are loaded such that the first specified in the command string to Link-11 is highest in core.

A sample load map is shown below.

```
LOAD MAP

TRANSFER ADDRESS:  Ø37434
LOW LIMIT:  Ø374Ø6
HIGH LIMIT:  Ø3746Ø
**********


MODULE  MOD1
SECTION ENTRY     ADDRESS SIZE
<. ABS.>          ØØØØØØ  ØØØØØØ
<       >         Ø374Ø6  ØØØØ44
         X3       Ø3743Ø
         X4       Ø3744Ø
         X5       Ø37452
         X7       Ø3745Ø
**********

MODULE  MOD2
SECTION ENTRY     ADDRESS SIZE
<       >         Ø37452  ØØØØØ6
         X1       Ø37452
         X2       Ø37452
**********
**********

UNDEFINED REFERENCES
X6
```

2.4 CORE LIBRARY SYMBOL TABLE FILE

A Core Library is a set of routines and/or reserved areas whose components are known globally. When creating a Core Library, the Linker, in addition to the normal load module, outputs a Core Library Directory which is used to link other load modules to the components of the Library.

Construction of the Core Library by Link-11 includes generation of a Directory file containing the definitions of global symbols in the Library. This Core Library Directory file is then fed into the linking process for individual load modules so that the proper global linking can take place. The Directory file is the Link-11 symbol table from the Core Library Linking process.

All references within the Core Library are resolved when the Library is generated. That is, the linking process is from the load module to the Library. The Library cannot reference any undefined globals.

CHAPTER 3

OPERATING PROCEDURES

3.1 LOADING

Link-11 is built (linked) as explained in "Getting DOS on the Air". It is then loaded into core from the disk by typing the following Monitor command.

```
.RUN LINK      <CR>
```

NOTE

In the examples, typing the RETURN, LINE FEED, and SPACE keys are shown as <CR>, <LF>, and <SPACE>, respectively. Also, in the examples, program printout is underlined whereas user typed input is not.

When the Linker is in core and ready to accept the user's command, it prints the following three lines:

```
LINK-11  V00A

PASS1
#
```

The user can now type a command string after the # sign.

3.2 COMMAND STRING

Commands are typed in response to the number sign, #, printed by the Linker. The format of the command string adheres to the requirements of the DOS Command String Interpreter (CSI), as explained in the Disk Operating System Monitor, Programmer's Handbook.

The Linker's file specifications must appear in the following order:

```
#load module, load map, core library symbol table < object modules   <CR>
```

A null specification field signifies that the associated output is not desired. A complete input specification contains the following information:

```
dev:filnam.ext[uic]/s1:v/s2:v.../sn:v <CR>
```

The default values for each specification are noted below.

	dev	filnam	ext	uic
Load Module	*	**	LDA	This user
Map Output	*	none	MAP	This user
Object Module	*	none	OBJ	This user
Symbol Table	*	none	STB	This user

*system device or last device specified on this side of the < symbol

**the filename from the first input specification

If a syntactical error is detected in a command string, the Linker will print the command on the teleprinter up to and including the character in error, followed by a question mark, and then a line beginning with the input request character #. The user can then type the entire command correctly.

3.2.1 Switches

There are ten switches associated with the Linker:

/T	Top
/B	Bottom
/U	Undefined globals
/OD	ODT
/TA	Tapes
/CC	Concatenate
/TR	Transfer address
/E	Exit
/L	Library
/RT	Real-Time header (see Chapter 7)

The letter(s) representing each switch is always preceded by the slash symbol. Switches are not allowed in the output fields of a command string.

3.2.1.1 Top and Bottom Switches

The T and B switches are used to control the placement or relocation of the object program. When neither switch is specified, Link-11 will link the object programs to the top of available core, i.e., immediately below the Absolute and Bootstrap Loaders.

The T switch (top) can be specified with any of the input file specifications. It must be in the following format:

```
/T:n
```

where n is an unsigned octal number which defines a new top address.

The B switch (bottom) can be specified with any of the input file specifications. It must be in the following format:

```
/B:n
```

where n is an unsigned octal number which defines the bottom address of the object program. The B switch causes a top address to be calculated so that the lowest address in the program will be at location n.

If more than one T or B switch is specified during the creation of a load module, the last T or B switch specification is used. When the load module creation is either finished or aborted, the default top reverts to its original value, i.e., the top of core of the installation.

CAUTION

The top or bottom value must be an unsigned even octal number. Link-11 detects if the value is odd and gives an error message.

3.2.1.2 Undefined Globals Switch

The U switch is used to obtain a teleprinter listing of all globals still found to be undefined at the time the switch is used. The listed globals will apply only to those files specified prior to the U switch request, including the specification in which the U itself appears.

The U switch can be specified with any or all input file specifications. Its format is:

```
/U
```

This switch does not require a value.

3.2.1.3 Tapes Switch

The TA switch is used to specify the number of tapes to be read in during the linking process. Its format is:

```
/TA:n
```

where n is an unsigned decimal number, the number of tapes to be read. If n is too large, the user must load some dummy modules or put a blank tape in the reader for each extra request on both pass 1 and pass 2.

In response to this switch, the Linker requests the DOS Monitor to read n tapes from the specified input device. The Monitor will print:

```
A00 2    063320
$
```

whenever the device is not ready. The message A002 means that device 063320 (high-speed paper tape reader, represented in Radix-50) is not ready.

The user should place the next tape in the reader and type CO<CR> in response to the $ printed by the Monitor to continue the linking process. The procedure above is repeated for each tape to be linked.

After pass 1, the Linker will print PASS 2 and the procedure above is repeated, producing the load module. After completing pass 2, control remains in the Linker for more link requests.

3.2.1.4 Concatenate Switch

The CC switch is used to indicate that the file was formed (by PIP or the FORTRAN compiler) as a concatenation of several object modules. This switch must be placed with an input file specification. Its format is:

```
/CC
```

This switch does not have a value.

3.2.1.5 ODT Switch

The OD switch is used to link ODT with your object modules. It identifies the associated input file as ODT for Transfer address purposes. /OD appearing by itself in an input file specification is equivalent to

```
ODT.OBJ[1,1]/OD
```

3.2.1.6 Transfer Address Switch

The TR switch can appear with any input file specification. It can be used with no value, or with an octal number or global symbol as its value.

When the TR switch has no value, it indicates that the Linker should take the transfer address (even or odd) of the first object module in the file as the transfer address of the load module. Its format is:

```
/TR
```

When it has an octal number as its value, it indicates that the value is the transfer address of the load module. Its format is:

```
/TR:n
```

When it has a global symbol as its value, it indicates that the value of the global symbol is the transfer address of the load module. Its format is:

```
/TR:xxxxxx
```

When the value is a nonexistent symbol, the transfer address is set to 1.

3.2.1.7 Exit Switch

The E switch should appear with the last input file specification. It indicates the end of input.

3.2.1.8 Library Switch

The L switch is used to indicate that the file is a library. It must appear in an input file specification, if the specification contains a user library. The L switch does not require a value. Its format is:

```
/L
```

3.2.1.9 General Notes on Switches

If a switch appears by itself as a specification (e.g., , , ,/CC), it takes the default device and a null file name. Thus, the linking process will be aborted if the default device is file structured.

There are thirty words allowed for switches per input specification. If more are requested, it will result in a S203 error message. The following conditions also result in a S203 error message:

a. If a switch requires a value and if none appears or more than one appears.

b. If a switch does not require a value and some value is given.

A comprehensive summary of all switches appears in Chapter 5.

3.3 LIBRARY SEARCHES

3.3.1 User Libraries

Object modules from the named user libraries built by the Libr-11 Librarian will be relocated selectively and linked by the Linker. The object modules in the Libraries have to be ordered; only forward references are allowed.

The libraries are input to the Linker as any other input file. The L switch in the input file specification indicates that this particular file is a library. It has the following format:

```
dev:libnam.ext/L
```

For example, the user could type the following command string to the Linker:

```
#TASKØ1.LDA,LP:<MAIN.OBJ,MEASUR.LIB/L/E
```

Program MAIN.OBJ would be read in from the disk as the first input file. Any undefined symbols generated by program MAIN.OBJ can be satisfied by the library MEASUR.LIB specified in the second input file. The load module, TASKØ1.LDA would be put on the disk, and a load map would go to the line printer.

3.3.2 Monitor Library

At the end of pass 1, the Monitor library is searched for Monitor routines which were declared as globals in the user program. Satisfying these globals means that the Linker passes the EMT trap numbers of the found routines (in the COMD) to the Monitor so that at load time the requested routines are made resident with the user program.

The user libraries are searched first and the Monitor library is searched last.

3.4 PROGRAMMING NOTES AND CAUTIONS

3.4.1 Programming Notes

1. No switch (except /OD) can appear alone by itself in an input specification (i.e., ,/U, and ,/E are illegal ,dev:filename.ext/U, and dev filename.ext/E are legal).

2. There are several ways to link ODT-11R.

   a. dev:outfil.ext,mapdev:<dev:test,dev:ODT11R.OBJ/OD/E

      (The OD switch is set)

      allows the Monitor command BE to start the test program, and the Monitor command OD to start ODT (assuming in both cases that the Monitor command GET OUTFIL.EXT precedes).

   b. dev:outfil.ext,mapdev:<dev:ODT11R.OBJ,dev:test/E

      (no OD switch)

      allows BE to begin ODT. OD will have no effect.

   c. dev:outfil.ext,mapdev:<dev:ODT11R.OBJ/E

      allows ODT to be run by itself with the BE command.

3. When using the /OD switch:

default device	--	as usual, disk
default filename	--	ODT
default extension	--	as usual, OBJ
secondary UIC	--	1,1

   If ODT.OBJ is on either the system or user disk area, the input file specification for ODT in the CSI reduces to:

   /OD

   for example:

   <TEST,/OD)

   The reduced form, /OD, cannot be the first input file specification. However, the following is valid:

   sysdev:ODT[1,1]/OD,

4. a. When creating object modules with the DOS Assembler or when transferring object modules with PIP, always specify in the output file specification a file extension in order to be able to link it with Link-11, which assumes extension .OBJ when none is specified.

   b. Always specify a file extension for the output file of Link-11; specify the same file extension when issuing the Monitor commands RUN or GET. For a detailed description of the search order see the Disk Operating System Monitor Programmer's Handbook (DEC-11-MWDB-D) section 3.3.2.1.

5. If an output error occurs (load module, load map, or core library), the dev:filename.ext part of the error message is not printed.

6. The Linker will not link libraries from paper tape.

7. The /OD and /TR switches are mutually exclusive.

8. If the user has a library which contains backward references, put the library file into the command string to the Linker repeatedly until all references are satisfied. For example, given the four object modules:

   PA, PB, PC, PD

Object Module PA:

```
.TITLE PA
.GLOBL B
.END
```

Object Module PB:

```
.TITLE PB
.GLOBL B,C
B:1
.END
```

Object Module PC:

```
.TITLE PC
.GLOBL C
C:1
.END
```

Object Module PD:

```
.TITLE PD
.END
```

where PA references PB, and PB references PC.

If we create three files F1.OBJ, F2.OBJ, and F3.OBJ where:

(a) file F1.OBJ contains object module PA,

(b) file F2.OBJ contains a library in which PB is first and PC is second, and

(c) file F3.OBJ contains object module PD

then the four programs can be linked with the following command strings:

```
#OUT,MAP<F1,F2/L/U
```

There were no undefined references left, so conclude the linking process by typing the command string:

```
#F3/E
```

If file F2.OBJ contains a library in which object module PC is first and PB is second, then after the command string:

```
#OUT,MAP<F1,F2/L/U
```

the following message will be printed:

```
UNDEFINED REFERENCES
C
```

Now issue the second command string:

```
#F2/L/U/
```

There will be no more undefined references left this time, so the linking process can be concluded by typing the third (final) command string:

```
#F3/E
```

3.4.2 Cautions

If the user means to type:

```
PP:,LP:<PR:/E
```

but accidentally types:

```
PP:,LP<PR:/E
```

the load map (an ASCII file) will be punched on the paper tape before the load module (a binary file). The Linker will not detect the error since the erroneous string is a legal one (i.e., output file LP.MAP to default device, PP:), thus the load module cannot be loaded (since there is an ASCII file in front of it).

There is a fair amount of blank tape between the load map and the load module, so either separate them or relink with the correct command string.

CHAPTER 4

ERROR HANDLING AND MESSAGES

4.1 RESTARTING

The user can restart Link-11 at any time by typing

CTRL/C RESTART or CTRL/C BEGIN

which causes Link-11 to re-identify itself and print #.

If the above sequence of keys (CTRL/C, RE(START) or BE(BEGIN), RETURN key) is typed while a load map is being printed, the load map will be aborted and the Linker will continue.

4.2 WARNING ERROR MESSAGES

The following types of warnings are printed by Link-11.

a. Non-unique object module name. This error is detected during pass 1 and results in the message:

XXXXXX

W3ØØ ØØØØØØ

where XXXXXX is the non-unique object module name. The module is rejected. The Linker will then continue processing the remaining unserviced file specifications.

Exception:

When linking a FORTRAN overlay file to a FORTRAN resident section with the aid of the resident symbol table, both contain an object module with the title "MAIN." . The Linker does not print a message and does not reject the second module with the name "MAIN." . Instead, the Linker changes the name of the resident program "MAIN." to "$$$N." in the resident section symbol table to facilitate processing.

b. A byte relocation error. The Linker will try to relocate and link byte quantities. Failure is defined as the high byte of the relocated value (or the linked value) not being zero. In such a case, the value is truncated to 8 bits and the following message is printed:

W3Ø1 XXXXXX

where XXXXXX is the absolute address where the byte relocation error occurred. The Linker automatically continues.

c. Multiply-defined globals. This results during pass 1, causing the error message:

XXXXXX, YYYYYY

W3Ø2 ØØØØØØ

where YYYYYY is the symbol which is multiply defined by object module XXXXXX. The second definition is ignored and the Linker continues.

Exception:

When linking a FORTRAN overlay file to a FORTRAN resident section with the aid of the resident symbol table, both contain the global declaration "MAIN". The Linker does not print a message and the second definition is not ignored. Instead, the Linker changes the global declaration "MAIN" to $$$N. in the resident section symbol table.

d. Undefined globals in the load module. This results at the end of pass 1, causing the error message:

W322 ØØØØØØ

meaning that there were some global symbols left undefined at the end of pass 1. The Linker then continues.

e. Named CSECT Size in object module to be linked is larger than same named CSECT size in core library symbol table.

If a named CSECT in an object module has a larger size than the one in the core library, then the Linker reports a warning message on the Teletype:

W323 ØØØØØØ

ignores the surplus information, and continues processing the object modules.

f. Named CSECT in an object module contains more global symbols than the same named CSECT in the core library symbol table.

If a named CSECT in an object module contains more globals than the same named CSECT in a core library, then the Linker reports a warning message on the Teletype:

W324 ØØØØØØ

ignores the surplus information, and continues processing the object modules.

4.3 ACTION REQUEST MESSAGES

If the object modules are not read in from paper tape in the same order on pass 2 as on pass 1, the Linker will indicate which module should be loaded next by printing:

```
XXXXXX
AØØ6   ØØØØØØ
```

meaning, load object module XXXXXX next.

When the message appears on the teleprinter, the Linker suspends the creation of the load module. The tape which is in the paper tape reader is ignored. The Linker waits for the user to remove the incorrect tape and place the correct tape in the reader.

When the user is done he types CO<CR> and the Linker reads in the tape and resumes the creation of the load module.

If input comes from a file-structured device, this action request does not apply.

4.4 FATAL LINKING ERROR MESSAGES

When Link-11 detects a fatal error condition during the linking process, it prints an appropriate error message and then restarts itself. Fatal error messages and their meanings are listed below.

Error Message and Format	Meaning
S202 ØØØxxx dev:file.ext	End of data or device error on .WRITE or .READ request. ØØØxxx = error status byte.
S203	Illegal switch, or too many switches, or illegal switch value, or switch value not given, or switch in an output field.
S204	Too many output file specifications.

Error Message and Format	Meaning
S206	Input file not specified in command string.
S207 xxxxxx	*End of data or device error on .TRAN request. xxxxxx = .TRAN Block FUNCTION/STATUS word.
S210 dev:file.ext	*Unrecognized symbol table entry was found in the indicated file.
S211 dev:file.ext	*A relocation command (RLD) references a global name which cannot be found in the symbol table of the indicated file.
S212 dev:file.ext	*A relocation command (RLD) contains a location counter modification command which is not last in the indicated file.
S213 dev:file.ext	An object module does not start with a global symbol directory (GSD) command in the indicated file. This message usually occurs when an input file is not an object module (but is probably a source or load module), or when the /L switch is not attached to a library input file specification.
S214 dev:file.ext	*The first entry in a global symbol directory is not the module name of the file indicated.
S215 dev:file.ext	*A relocation command references a section name that cannot be found in the file indicated.
S216	*The load module starting address (TRA) references a nonexistent module name.
S217	*The load module starting address (TRA) references a nonexistent section name.
S220	*An internal jump table index is out of range.
module,symbol S223	Symbol table overflow. Insufficient space in symbol table for symbol of object module indicated.
S223	No more space for the command string interpreter (CSI) input buffer, or Monitor's file manager routine, or for Monitor's library search buffer.
S225	Program too large or top linked too low -- the program has been linked below zero in core.
S226	An angle bracket (<) is present in a line other than the first.
S243	The angle bracket (<) is missing from the first command string line.

*These errors are caused by a Linker or Assembler internal failure.

CHAPTER 5

SUMMARY OF LINK-11 SWITCHES

Name	Symbol	Value	Format	Function
Top	T	n (octal number)	/T:n	n becomes top of load module
Bottom	B	n (octal number)	/B:n	n becomes bottom of load module
Undefined Globals	U	none	/U	List undefined globals on teleprinter
ODT	OD	none	/OD	This file is ODT
Tapes	TA	n (decimal number)	/TA:n	There are n tapes to be read
Concatenate	CC	none	/CC	This file contains concatenated object modules
Transfer Address	TR	none	/TR	Take the transfer address of the first object module of this file as the transfer address
	TR	n (octal number)	/TR:n	Take the octal number as the transfer address
	TR	global symbol	/TR:xxxxxx	Take the value of the global symbol as the transfer address
Library	L	none	/L	This file is a library
Exit	E	none	/E	Exit from current pass; end of command string for this link
Real-Time	RT	g (global symbol) n1 - n4 (decimal numbers)	/RT:g :n1 :n2 :n3 :n4	Specifies or modifies RSX11 real-time headers. g = task name n1 = priority n2 = call number n3 = maximum time n4 = stack size (see Chapter 7)

CHAPTER 6

PROGRAM OVERLAY FACILITY

6.1 PURPOSE OF AN OVERLAY FACILITY

The PDP-11 DOS program overlay facility enables the user to have virtually unlimited core space for his program. A program using the overlay facility can be much larger than would normally fit in the available core space, since portions of the program, called overlays, reside on the backup storage (that is, disk, DECtape, or magtape) and are brought into core as needed.

Version 1 of the overlay facility contains a subset of the features available in the complete overlay facility package to be released at a later date. Overlays are useful when handling large FORTRAN programs and can also be used with assembly language programs.

The run time supervisor (which is linked to the resident section of the user program) of the overlay facility supplied with the PDP-11 FORTRAN system is the subroutine LINK in the FORTRAN Library (FORLIB.OBJ or EAELIB.OBJ, version VØ15A). The LINK subroutine currently loads other support routines (RUN and LOAD) with a total size of $620_{1Ø}$ words of core.

In the next release of DOS, the support routines will be replaced by a DOS RUN EMT. This will reduce the total run time supervisor to approximately $15Ø_{1Ø}$ words.

The overlay system which the user constructs from his completed program is composed of a core resident section, a core resident overlay area, and overlay files stored on backup storage.

For example, a FORTRAN system could contain a resident section (including the run time supervisor) and maintain an overlay area into which any number of overlay files could be moved as required.

LINK-11 (VØØ7A) is used to link the core resident section of the user program to the Library routines required (including the overlay supervisor). LINK-11 is also used to create the various overlay files to be used with the resident section.

Overlays communicate through blank or labelled COMMON in the resident area or through the contents of general registers RØ through R4 and the stack. Overlay files can use any core resident routines (user generated or acquired from the FORTRAN Library).

The user is advised to organize the overlay strategy for his program prior to program development to provide for COMMON usage, format conversions, and core residency requirements.

6.2 COMMUNICATION AMONG RESIDENT AND OVERLAY ROUTINES

Overlays and the resident section communicate through blank and labelled COMMON areas (.CSECT's for assembly language programs) in the core resident area. Data can also be passed between overlays in the general registers RØ through R4 and the stack.

6.3 FORMAT CONVERSIONS

Any format conversions not needed in the resident section but required by overlay sections must be forcibly loaded into the resident section. This can be done by declaring the appropriate globals in an assembly language routine or inserting dummy FORMAT statements in the resident main program for all those format conversions needed in the overlays and not required in the resident section.

For example, assume I and L format conversions are needed for READ and I and E format conversions are needed for WRITE. An assembly language routine such as the following could be written:

```
.TITLE DUMMY
.GLOBL $LCI,$ICI,$ICO,$DCO
.END
```

where $LCI performs the L conversions for READ, $ICI performs the I conversions for READ, $ICO performs the I conversions for WRITE, and $DCO performs the E conversion for WRITE.

Alternatively, dummy FORMAT statements can be supplied in the resident main program to force linking of these routines. (If this is done, a message is printed at compile time indicating that there is non-executable code in the program.)

```
      LOGICAL L
        .
        .
        .
      GO TO 1ØØØ
      READ (6,1ØØ) I,L
  1ØØ FORMAT (I1,L1)
      WRITE (6,1Ø1) I,E
  1Ø1 FORMAT (I1,E6.Ø)
 1ØØØ CONTINUE
        .
        .
        .
```

The first method causes the four format conversion packages to be linked to the resident program section, while the second method causes the resident to carry the overhead of the four format conversion packages plus the FORTRAN READ/WRITE processor and associated routines.

Two other possibilities are either to perform all I/O in the resident program only, or to perform all I/O in only one of the overlay files.

If those format conversions which are needed in the overlays and not required in the resident section are not forcibly loaded into the resident section, the FORTRAN system causes the linking of dummy conversion routines. Global requests in the overlay files are then linked to the resident dummy routines and at execution time results in the fatal error message:

```
F3Ø ØØØØØØ
```

The following table is useful in building overlay systems. If any conversion not needed by the resident is required in an overlay, then the corresponding global must be declared in the resident section.

TABLE 6.1

FORMAT CONVERSION PACKAGES

Globals in Package	Conversion Performed	Length of Package in Decimal Words
$DCO, $ECO, $FCO, $GCO	Output conversions D,E,F,G	469
$ICO, $OCO	Output conversions, I, O	93
$LCO	Output conversions L	31
$DCI, $RCI	Input conversions D,E,F,G	384
$ICI, $OCI	Input conversions I,O	85
$LCI	Input conversions L	31

6.4 LINK, THE RUN-TIME OVERLAY SUPERVISOR

The subroutine LINK is the run-time overlay supervisor. It has two entry points: LINK and RETURN.

The subroutine LINK performs the following functions for the user when entered at LINK:

a. initializes the traceback origin to the new overlay file to allow FORTRAN error tracking via the traceback feature.

b. if called from the resident section saves general registers RØ through R5 and saves the return address to the resident section for use with the next CALL RETURN.

c. brings the overlay file (designated by the argument to CALL LINK) into core and moves the stack below the new overlay file if the stack would be penetrated by that overlay.

The subroutine LINK performs the following functions for the user when entered at RETURN:

a. restores general registers RØ through R5.

b. transfers control following the last CALL LINK executed from the resident section.

The LINK subroutine contains the global declaration:

```
.GLOBL $OTSV
```

where $OTSV is the pointer to the FORTRAN impure area. This forces the impure area into the core resident section so that user I/O can be continued across overlays.

6.5 CALLING AN OVERLAY FILE

The resident section and overlay files are entered into the system as load modules. An overlay is requested by a

```
CALL LINK ('A')
```

statement, where A is the file specification of the overlay file to be called into core. All overlays are entered through their main program, not through any subroutines.

The argument of the overlay call is enclosed in single quotes since the Linker relies on a terminating null character inserted by the FORTRAN Compiler at the end of such a string. The maximum length of the argument is 25 decimal bytes (stored one character per byte).

The call can be written as follows:

```
CALL LINK ('dev:file.ext')
```

Due to the DOS default conventions, the call can normally be expressed as:

```
CALL LINK('file')
```

with the system disk the assumed device and .LDA the assumed extension for a load module.

Whenever the statement:

```
CALL RETURN
```

is encountered, control transfers to the instruction following the last CALL LINK executed from the resident section.

The first CALL LINK statement is issued from somewhere in the core resident section of the user program, bringing into core the first overlay file. Subsequent CALL LINK statements can be issued from the main programs in the various overlays. Each CALL LINK brings into core the specified overlay file.

CALL LINK can be issued from anywhere in the core resident section (main program or subroutines). CALL LINK and CALL RETURN can be issued only from the main program in an overlay and not from a subroutine within an overlay.

There are four allowable control paths between resident and overlay load modules:

a. resident to overlay via CALL LINK
b. overlay to overlay via CALL LINK
c. overlay to resident via CALL RETURN
d. overlay utilizing resident subroutines.

Figure 6-1 illustrates these transfer paths.

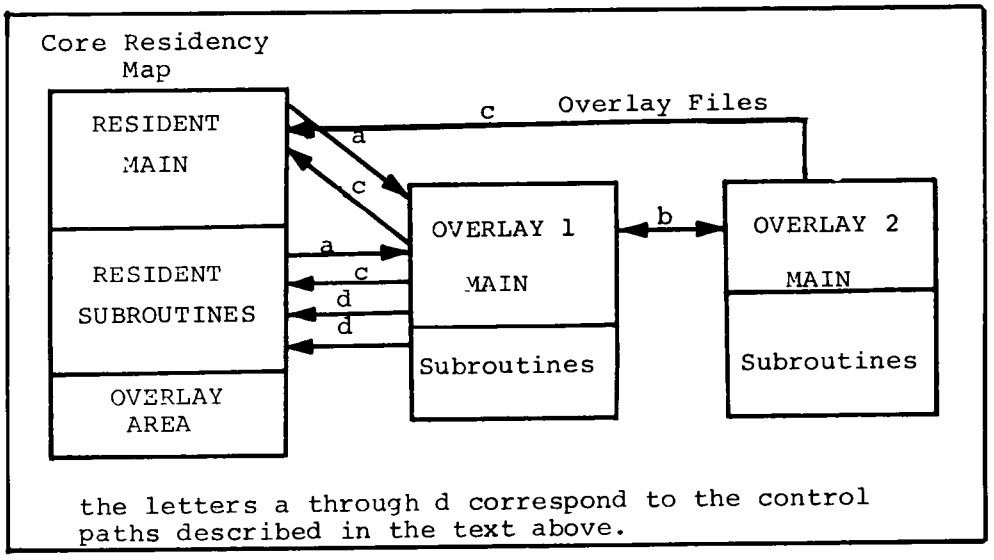


Figure 6-1
TRANSFER PATHS BETWEEN RESIDENT AND OVERLAY PROGRAMS

6.5.1 Overlay Transfer Paths

CALL LINK may be issued from anywhere in the resident section. However, in an overlay, CALL LINK and CALL RETURN can be issued only from the main program.

Resident subroutines can be called from anywhere in an overlay file (main program or subroutine). However, there is a restriction that an overlay must not call any resident routine which contains (or calls another routine which contains) a CALL LINK.

Violation of these control path restrictions causes the overlay system to be corrupted, resulting in stack overflow or incorrect subroutine returns.

6.5.2 Search for Overlay Files

When a CALL LINK is performed, the overlay file is searched under the specified device, filename, extension, and current user identification code (UIC). If not found, the overlay is searched under the specified device, filename, extension, and UIC 1,1.

If the overlay file is not found and an extension was specified, the search terminates at this point as a failure and a message is printed.

If the file is not found at this point and no extension was specified, the overlay is searched under the specified device, filename, extension .LDA, and the current UIC. Lastly, an attempt is made to find the overlay under the specified device, filename, extension .LDA, and UIC 1,1. If the overlay is still not found, the search ends as a failure.

If the search has failed, the DOS Monitor prints the message:

```
FØ12 XXXXXX
```

where the additional information word contains the request address (which is within the run time supervisor).

6.6 OPERATING PROCEDURES

LINK-11 (VØØ7A) is used to create the core resident and overlay load modules. The procedures resemble the normal operation of LINK-11.

6.6.1 Creating the Core Resident Module

The command string to LINK-11 when creating the core resident module has the explicit requirements that:

a. the symbol table (third output field of command string) must be specified as output, and

b. the FORTRAN Library must be specified as the last input file.

The format of the Linker command string is as follows:

```
dev:RES,dev:MAP,dev:ST<dev:RES1,...subroutines,FORLIB[1,1]/L/E
```

where:

dev:RES	is the file specification for the user's core resident section load module.
dev:MAP	is the load map of the resident program, which is an optional output file.
dev:ST	is the global symbol table of the resident program load module.
dev:RES1	is the file specification for the user's core resident main program.
...subroutines	indicates the user subroutines to be linked to the main program.
FORLIB[1,1]/L	the FORTRAN Library contains the overlay facility supervisor and other Library routines which must be linked to the user's resident section. /L switch specifies that the file FORLIB is a Library file.
/E	indicates the end of the command string to the Linker. This is followed by the RETURN key.

6.6.2 Creating the Overlay Files

The command strings to LINK-11 used when creating the various overlay files have the explicit requirements that:

a. the symbol table of the resident section is the first input file specification for each overlay file, and

b. the top address for each overlay file must be explicitly specified as LO-2, where LO is the low limit of the resident section acquired from the resident load map.

c. the FORTRAN Library is linked to each overlay.

The format of these commands to the Linker is as follows:

```
dev:OVER,dev:MAP<dev:ST.STB,dev:OVER1,...subroutines,FORLIB[1,1]/L/T:n/E
```

where:

dev:OVER	is the file specification for the overlay load module. This specification must correspond exactly to the one used as the argument to CALL LINK when this overlay is wanted.
dev:MAP	is the load map of the overlay, which is an optional output file.
dev:ST.STB	is the symbol table of the resident section (must be the first input file specification).
dev:OVER1	is the file specification for the overlay main program.
...subroutines	indicates the user subroutines to be linked to this particular overlay, if any.
FORLIB[1,1]/L	the FORTRAN Library must be linked to each overlay. The /L switch specifies the file FORLIB as a library file.
/T:n	is the switch to indicate the top of the overlay file; n is equivalent to the address LO-2 as explained previously.
/E	indicates the end of the command string to the Linker. This is followed by the RETURN key.

Since the default device for load modules is always the system device and the default extension is .LDA, the load module file specification can be simplified for both resident and overlay file creation as follows:

```
RES,dev:MAP,dev:ST<dev:RES1,...subroutines,FORLIB[1,1]/L/E
OVER,dev:MAP<dev:ST.STB,dev:OVER1,...subroutines,FORLIB[1,1]/L/T:n/E
```

The structure of the program overlay facility allows modifications to be made to overlay load modules without relinking the resident and other overlay load modules.

6.7 ERROR PROCEDURES AND MESSAGES

Error handling during the creation of the overlay system is performed as for LINK-11. At run time there are three fatal errors which are concerned with the overlay facility. These are as follows:

Error Code		Explanation
F275	XXXXXX	Argument to the LINK subroutine is syntactically incorrect or too long.
F276	XXXXXX	Transfer address of the overlay file was not specified. For FORTRAN overlays, no main program was included in the overlay file.
F277	XXXXXX	Overlay file could not be brought into core because it would overlay the resident section. This is the only error condition which generates the F277 code.

XXXXXX is the additional information word which contains the address in the user code following the offending CALL LINK.

After a F275 error message (where the additional information is the address following the offending CALL LINK), identify the offending CALL LINK and search for one of the following error conditions in the argument of CALL LINK:

a. syntactically incorrect file specification,
b. multiple file specifications,
c. illegal switch specification, or
d. file specification longer than 25 decimal characters.

When the error is isolated, correct it. Reassemble or recompile the object module in which the error occurred. If the error occurred in the resident section, relink the entire overlay system. If the error occurred in an overlay file, then relink only the particular overlay file with the aid of the Core Library symbol table of the resident section, and the specified top switch (resident lower limit-2) and the FORTRAN Library.

After a F276 error message, determine the offending CALL LINK as for F275. If the overlay file is a FORTRAN program, the error was probably caused by not including a main program in the overlay file. If the overlay file was composed of assembly language modules, then the error was caused by not specifying a transfer address. Follow relinking instructions described under F275.

After a F277 error message, determine the offending CALL LINK as for F275. This error results when the overlay would have overlaid part of the resident section. To correct the error condition, the offending overlay must be relinked with a top of (low limit of the resident-2). See relinking instructions under F275.

6.8 ASSEMBLY LANGUAGE OVERLAYS

The creation of the overlay system (resident and overlay load modules) is the same for assembly language programs as for FORTRAN programs..

The assembly language calling sequence for the LINK subroutine is as follows (the Standard FORTRAN calling sequence):

```
        R5=%5
        JSR R5,LINK
        BR .+4                  ;RETURN ADDRESS
        .WORD  NAME             ;POINTER TO ARGUMENT
         .
         .
         .
NAME:   .ASCII  /DEV:FILE/      ;ARGUMENT
        .BYTE Ø                 ;ASCII CHARACTER STRING OF OVERLAY
        .EVEN                   ;LOAD MODULE FILE SPECIFICATION
                                ;TERMINATING NULL BYTE
```

The assembly language calling sequence to return to the resident program is as follows (the standard FORTRAN calling sequence):

```
JSR  R5,RETURN
BR  .+2
 .
 .
 .
```

6.8.1 Global Declarations

In all assembly language programs that call LINK or RETURN, the symbols LINK and RETURN must be declared as globals.

```
.GLOBL LINK, RETURN  ;EXTERNAL REFERENCES
```

6.8.2 Stack Usage

Assembly language resident code should not reset the stack at execution time to be at a lower address than set by the Monitor at loading time. The Monitor considers the stack as starting at the address set at loading time and ending at the contents of register 6 and uses this information to first decide if the stack must be moved and then to move it below an incoming overlay.

The stack cannot be addressed absolutely (e.g., pointers to the stack cannot be stored on the stack itself).

Exit from an overlay should be made only through the main program of that overlay, otherwise the return addresses from any nested subroutines are lost on the stack. (See Section 6.5.1.)

While executing FORTRAN programs, the stack is only used to store the return addresses from nested subroutines. (The latest entry on the stack is the old contents of register 5 and the return address to the last subroutine call is in register 5.)

With these restrictions in mind, the stack can be used by the assembly language program.

6.8.3 Register Usage

Before exiting from the resident program to an overlay, general registers RØ through R5 are saved by the LINK subroutine. On reentering the resident program from an overlay, registers RØ through R5 are restored by the RETURN subroutine.

The register contents are not disturbed by LINK; therefore, arguments can be passed between overlays through the registers RØ - R4.

With these points in mind, the general registers can be used by the assembly language program.

6.8.4 COMMON Communication

COMMON communication between a FORTRAN main program and an assembly language subroutine is shown below. In the example, the variable I of blank COMMON is input through the keyboard. First the two low order bits then the five low order bits of I are cleared by the mask in variable M of labelled COMMON through the assembly language subroutine MASK.

Note that two words are allocated for integers in the assembly language subroutine MASK in order to be compatible with FORTRAN storage allocation.

The following is the FORTRAN main program:

```
      COMMON I,J
      COMMON /X/L,M
5     WRITE(6,1)
1     FORMAT(1H ,7H INPUT )
C     INPUT VARIABLE I OF BLANK COMMON
C     FROM KEYBOARD
      READ(6,2) I
2     FORMAT(I5)
C     PREPARE THE MASK FOR
      THE TWO LOW ORDER BITS
      M=3
      CALL MASK
      WRITE(6,3) I
3     FORMAT(1H ,2HI=,I5)
C     PREPARE THE MASK FOR BITS
      Ø THRU 4 (31(1Ø) = 37(8)).
      M=31
      CALL MASK
      WRITE(6,4) I
4     FORMAT(1H ,2HI=,I5)
      GO TO 5
      END
```

The assembly language subroutine MASK follows:

```
        .TITLE  MASK
        .GLOBL  MASK

;CLEAR THE 1ST POSITION OF BLANK
;COMMON BY THE MASK OF THE 2ND POSITION
:OF NAMED COMMON, X.
MASK:
        R5=%5
        .CSECT  .$$$$.
        S=.+Ø
        T=.+4
        .=.+1Ø
        .CSECT X
        A=.+Ø
        B=.+4
        .=.+1Ø
        .CSECT
        BIC     R,S
        RTS     R5
        .END
```

6.9 PROGRAMMING NOTES

"Run-only" protected files cannot be accessed by users other than the file owner. If such access is attempted through the overlay supervisor, the Monitor prints a fatal error message

F Ø 1 2 XXXXXX

where XXXXXX is the request address in library subroutine RUN. This condition will be corrected when the RUN EMT is implemented.

The DOS Monitor command BEGIN cannot be used to start FORTRAN programs.

The DOS Monitor ASSIGN command is effective while using overlays. For example, where overlay files are specified through the argument of the LINK subroutine to be obtained from the disk at execution time and those overlay files are moved onto DECtape unit 1, the command:

```
AS DT1:
```

causes all overlay files to be obtained from DECtape unit 1. Where the overlay files were originally specified as being on various devices, an assignment made to a particular device causes all overlay files to be searched on that device.

At present in VØØ7A, you can reinitialize COMMON in the resident from an overlay via BLOCK DATA subprograms in FORTRAN or via a .WORD directive in assembly language. However, the use of this feature is discouraged because in future versions core image (contiguous) file overlays will not accommodate this feature, and the overlay file will be refreshed upon call only if it is not in core already.

When an AØØ3 message appears during program execution, it may be because the link subroutine argument was not enclosed within quotes.

When an FØ1Ø message appears during a linking session, it could have been because you did not specify a device for the third output field when LP: was specified as the second output field.

6.10 A SUMMARY OF THE MAJOR FEATURES OF THE OVERLAY FACILITY

The overlay system is composed of the resident section, a core resident overlay area and the overlay files. The user can specify the object modules to be included in the resident section or the overlay files at overlay build time. The resident section and overlay files are load modules. Overlays can use resident routines.

An overlay is requested via CALL LINK ('A'), where A is the file specification of the overlay which must contain a FORTRAN main program and, optionally, subroutines. CALL LINK ('A') can be issued from the resident section or from an overlay main program. CALL RETURN can be issued only from an overlay main program and returns control to the resident section.

Overlays communicate through blank and/or labelled COMMON (.CSECT's for the assembly language programmer) in the resident area. I/O can continue across overlays. Overlays can be modified without relinking the resident and other overlay load modules. Overlays are brought into core at each call.

6.11 PROGRAMMING EXAMPLE OF OVERLAY SYSTEM CREATION

The following pages contain an example of an overlay system which uses all the features of the PDP-11 overlay facility. A main resident program and resident subroutines are shown along with four short overlays.

Following the listing of the program source code is the disk directory before linking the overlay system, the command strings to the Linker, and the disk directory after linking the overlay system. Output from the run of the program is also shown.

The following is the source of the resident section, including the main program and subroutine.

```
C       THIS PROGRAM ILLUSTRATES THE USE OF OVERLAYS. IT CALCULATES
C       THE SQUARE ROOT OF SINE OF ALPHA.  THE ANGLE ALPHA
C       IS INPUT IN OVERLAY 1.  THE SINE OF ALPHA IS CALCULATED
C       IN OVERLAY 2.  THE SQUARE ROOT OF SINE ALPHA IS FOUND IN OVERLAY 3.
C       THE RESULTS ARE PRINTED IN OVERLAY 4.
C       LABELLED COMMON AREA FOR THE RESIDENT SECTION
C       AND FOR THE OVERLAYS.
        COMMON /X/A
C       BLANK COMMON AREA FOR THE RESIDENT SECTION
C       AND FOR THE OVERLAYS.
        COMMON B,C
        LOGICAL *1BUFF1(10),BUFF2(10),BUFF3(10),BUFF4(10)
        DIMENSION BUFF1(10),BUFF2(10),BUFF3(10),BUFF4(10)
        DATA BUFF1/'RE','SI','DE','NT',05015,'  ','  ','  ','  ','  '/
        DATA BUFF2/'LA','BE','L2',05015,'  ','  ','  ','  ','  ','  '/
        DATA BUFF3/'LA','BE','L3',05015,'  ','  ','  ','  ','  ','  '/
        DATA BUFF4/05015,'LA','RE','L4',05015,'  ','  ','  ','  ','  '/
C       CALL THE RESIDENT SUBROUTINE "PRINT" TO PRINT "RESIDENT"
C       TO INDICATE THAT EXECUTION STARTED IN THE RESIDENT SECTION.
1       CALL PRINT(BUFF1)
C       CALL THE RESIDENT SUBROUTINE "LINK" TO BRING THE FIRST OVERLAY
C       INTO CORE FROM DISK.  IN THE FIRST OVERLAY THE ANGLE ALPHA
C       IS INPUT FROM THE KEYBOARD WITH A FORMAT OF F5.2.
        CALL LINK('OVL1')
C       RETURNED TO RESIDENT AT LABEL 2 FROM OVERLAY 1 VIA
C       A CALL RETURN.
2       CALL PRINT(BUFF2)
C       CALL THE RESIDENT SUBROUTINE "LINK" TO BRING INTO CORE FROM
C       DISK THE SECOND OVERLAY ON TOP OF THE FIRST OVERLAY.
C       IN THE SECOND OVERLAY THE SINE OF THE ANGLE ALPHA IS
C       CALCULATED.
```

```
      CALL LINK('OVL2')
C     RETURNED TO RESIDENT AT LABEL 3 FROM OVERLAY 2 VIA A
C     CALL RETURN.
3     CALL PRINT(BUFF3)
C     CALL THE RESIDENT SUBROUTINE "LINK" TO BRING INTO CORE FROM
C     DISK THE THIRD OVERLAY ON TOP OF THE SECOND OVERLAY.  IN
C     THE THIRD OVERLAY THE SQUARE ROOT OF THE SINE OF THE ANGLE
C     ALPHA IS CALCULATED.
      CALL LINK('OVL3')
C     RETURNED TO RESIDENT AT LABEL 4 FROM OVERLAY 4 VIA A CALL RETURN
4     CALL PRINT(BUFF4)
      GO TO 1
      END



      SUBROUTINE PRINT (BUFF)
C     THIS IS A RESIDENT SUBROUTINE TO BE CALLED FROM THE RESIDENT
C     SECTION OR FROM THE OVERLAYS
C     THIS SUBROUTINE IS USED TO PRINT OUT TEXT ON THE TELETYPE,
C     MAINLY TO IDENTIFY THE SECTION WHICH IS BEING EXECUTED.
      LOGICAL *1BUFF(10)
      DIMENSION BUFF(10)
      WRITE(6,5) BUFF
5     FORMAT(1H ,12A1)
      RETURN
      END



      .TITLE FORCE
;     THE SOLE PURPOSE OF THIS ASSEMBLY LANGUAGE MODULE IS TO
;     FORCIBLY LOAD THE "F" FORMAT INPUT & OUTPUT CONVERSION
;     ROUTINES INTO THE RESIDENT AREA FOR USE BY OVERLAY 1 &
;     OVERLAY 4 RESPECTIVELY. THIS IS ACCOMPLISHED BY
;     DECLARING AS GLOBALS $DCI & $FCO. THESE TWO GLOBALS
;     WILL CAUSE THE LINKER TO LOAD FROM THE FORTRAN LIBRARY
;     THE TWO REQUIRED FORMAT CONVERSION ROUTINES.  NOTE!
;     THESE TWO CONVERSIONS ARE NOT REQUIRED IN THE RESIDENT.  BUT
;     IF THEY ARE NOT FORCIBLY LOADED IN THE RESIDENT THE
;     FORTRAN COMPILER WILL CAUSE THE LINKING OF TWO DUMMY CONVERSION
;     ROUTINES INSTEAD.  THEN GLOBAL REQUESTS IN THE OVERLAY LINKS
;     WILL BE LINKED TO THE DUMMY ROUTINES
;     AND AT EXECUTION TIME WILL RESULT
;     IN THE FATAL ERROR MESSAGE F030 000100.
      .GLOBL $FCO,$DCI
      .END



C     THIS IS OVERLAY 1.  IN THIS OVERLAY THE ANGLE ALPHA IS INPUT
C     FROM THE KEYBOARD WITH A FORMAT OF F5.2.
      COMMON /X/A
      LOGICAL *1BUFF5(13),BUFF6(10)
      DIMENSION BUFF5(14),BUFF6(10)
      DATA BUFF5/'OV','ER','LA','Y1',05015,'  ','  ','  ','  ','  '/
      DATA BUFF6/'IN','PU','T ',05015,05015,'  ','  ','  ','  ','  '/
C     CALL THE RESIDENT SUBROUTINE "PRINT" TO PRINT "OVERLAY1"
C     TO INDICATE THAT CURRENTLY OVERLAY 1 IS BEING EXECUTED.
      CALL PRINT(BUFF5)
C     CALL THE RESIDENT SUBROUTINE "PRINT" TO PRINT "INPUT"
```

```
C       TO INDICATE TO THE OPERATOR THAT THE COMPUTER IS WAITING
C       FOR THE VALUE OF THE ANGLE ALPHA FROM THE KEYBOARD.
        CALL PRINT(BUFF6)
C       READ INTO THE VARIABLE "A" IN THE LABELLED COMMON AREA THE
C       ANGLE ALPHA FROM THE KEYBOARD.
        READ(6,6) A
6       FORMAT(F5.2)
C       NOTE! THE CONTENTS OF BUFF6 IS BEING DYNAMICALLY CHANGED
C       AT RUN TIME. THIS WILL SHOW THAT OVERLAYS ARE COMPLETELY
C       REFRESHED ON THE NEXT CALL.THIS IMPLIES THAT NO DATA
C       COMMUNICATION CAN BE ACHIEVED BETWEEN OVERLAYS OTHER
C       THEN THROUGH RESIDENT COMMON.
        BUFF6(1)='O'
        BUFF6(2)='K'
        BUFF6(3)=' '
        BUFF6(4)=' '
        BUFF6(5)=' '
C       CALL THE RESIDENT SUBROUTINE "PRINT" TO PRINT "OK"
C       TO INDICATE TO THE OPERATOR THAT THE DATA HAVE BEEN
C       RECEIVED.
        CALL PRINT(BUFF6)
C       CALL "RETURN" TO REENTER THE RESIDENT ROUTINE WHERE THE LAST
C       CALL FOR "LINK" WAS ISSUED FROM THE RESIDENT.  IT IS
C       AT LABEL 2.
        CALL RETURN
        END



C       THIS IS OVERLAY 2.  IN THIS OVERLAY THE SINE OF THE ANGLE
C       ALPHA IS CALCULATED.
        COMMON /X/A
        COMMON B
        LOGICAL *1BUFF7(10)
        DIMENSION BUFF7(10)
        DATA BUFF7/'OV','ER','LA','Y2',05015,'  ','  ','  ','  ','  '/
C       CALL THE RESIDENT SUBROUTINE "PRINT" TO PRINT
C       "OVERLAY2" TO INDICATE THAT CURRENTLY OVERLAY2 IS BEING EXECUTED.
        CALL PRINT(BUFF7)
C       CALL THE FORTRAN LIBRARY ROUTINE "SIN", WHICH IS LINKED
C       TO THIS OVERLAY ONLY, TO CALCULATE THE SINE OF THE ANGLE
C       ALPHA.  THE ANGLE ALPHA IS IN VARIABLE "A" AND THE SINE OF
C       ALPHA WILL BE PUT IN VARIABLE "B".
C       "A" IS IN THE LABELLED COMMON AREA AND "B" IS IN THE
C       BLANK COMMON AREA.
        B=SIN(A)
C       CALL "RETURN" TO REENTER THE RESIDENT ROUTINE WHERE THE LAST
C       CALL FOR "LINK" WAS ISSUED FROM THE RESIDENT IT IS AT LABEL3.
        CALL RETURN
        END
```

```
C       THIS IS OVERLAY 3.  IN THIS OVERLAY THE SQUARE ROOT OF THE SINE
C       OF THE ANGLE ALPHA IS CALCULATED.
        COMMON B,C
        LOGICAL *1BUFF8(13)
        DIMENSION BUFF8(13)
        DATA BUFF8/'OV','ER','LA','Y3',05015,'  ','  ','  ','  ','  '/
C       CALL THE RESIDENT SUBROUTINE "PRINT" TO PRINT "OVERLAY3"
C       TO INDICATE THAT CURRENTLY OVERLAY 3 IS BEING EXECUTED.
        CALL PRINT(BUFF8)
C       CALL THE FORTRAN LIBRARY ROUTINE "SQRT", WHICH IS LINKED TO
C       THIS OVERLAY ONLY, TO CALCULATE THE SQUARE ROOT OF THE SINE OF THE
C       ANGLE ALPHA.  THE SINE OF ALPHA IS IN VARIABLE "B" AND THE
C       SQUARE ROOT OF IT WILL BE PUT IN VARIABLE "C".  BOTH ARE IN
C       THE BLANK COMMON AREA.
        C=SQRT(B)
C       CALL THE RESIDENT SUBROUTINE "LINK" TO BRING INTO CORE FROM
C       DISK THE FOURTH OVERLAY ON TOP OF THE THIRD OVERLAY, FROM
C       WHERE THE CALL WAS ISSUED.  IN THE FOURTH OVERLAY THE RESULTS
C       ARE GOING TO BE PRINTED ON THE TELETYPE.
        CALL LINK('OVL4')
        END



C       THIS IS OVERLAY 4. IN THIS OVERLAY THE RESULTS ARE PRINTED ON
C       THE TELETYPE.
        COMMON /X/A
        COMMON B,C
        LOGICAL *1BUFF9(13)
        DIMENSION BUFF9(13)
        DATA BUFF9/'OV','ER','LA','Y4',05015,'  ','  ','  ','  ','  '/
C       CALL THE RESIDENT SUBROUTINE "PRINT" TO PRINT "OVERLAY4" TO
C       INDICATE THAT CURRENTLY OVERLAY 4 IS BEING EXECUTED.
        CALL PRINT(BUFF9)
C       PRINT THE RESULTS ON THE TELETYPE.  ALL WITH THE FORMAT
C       OF F5.2.  THE ANGLE ALPHA IS IN VARIABLE "A", THE SINE OF
C       THE ANGLE ALPHA IS IN VARIABLE "B".  THE SQUARE ROOT OF
C       THE ANGLE ALPHA IS IN VARIABLE "C".
C       VARIABLE "A" IS IN THE LABELLED COMMON AREA AND VARIABLES
C       "B" AND "C" ARE IN THE BLANK COMMON AREA.
        WRITE(6,7) A,B,C
7       FORMAT(X,2HA=,F5.2,6X,7HSIN(A)=,F5.2,6X,12HSQRT SIN(A)=,F5.2)
C       CALL "RETURN" TO REENTER THE RESIDENT ROUTINE WHERE THE LAST
C       CALL FOR "LINK" WAS ISSUED FROM THE RESIDENT.  IT IS AT LABEL4.
        CALL RETURN
        END
```

The following output shows the disk directory before linking the overlay system:

```
$RU PIP

PIP-11  V05B
#/DI

DIRECTORY DF0: [200,200]

08-DEC-71

DUMMY .OBJ      2  08-DEC-71 <233>
PRINT .OBJ      7  08-DEC-71 <233>
RES   .OBJ     14  08-DEC-71 <233>
OVL1  .OBJ     16  08-DEC-71 <233>
OVL2  .OBJ      8  08-DEC-71 <233>
OVL3  .OBJ      8  08-DEC-71 <233>
OVL4  .OBJ     11  08-DEC-71 <233>

TOTL BLKS:     66
TOTL FILES:     7
```

The process of linking this overlay system is shown below: The commands to the Linker create the resident load module and the four overlay load modules. Note the presence of the resident symbol table, the FORTRAN Library file, and the top specifications while linking the overlay files.

```
$RU LINK

LINK-11  V05.7A

PASS 1
#RES,LP:,DF:RES<RES,PRINT,DUMMY,FORLIB[1,1]/L/E

PASS 2
```

```
LINK-11  V07A

PASS 1
#OVL1,LP:<RES.STB,OVL1/T:117620,FORLIB[1,1]/L/E

PASS 2
```

```
LINK-11  V07A

PASS 1
#OVL2,LP:<RES.STB,OVL2/T:117620,FORLIB[1,1]/L/E

PASS 2
```

```
LINK-11  V07A

PASS 1
#OVL3,LP:<RES.STB,OVL3/T:117620,FORLIB[1,1]/L/E

PASS 2
```

```
LINK-11  V07A

PASS 1
#OVL4,LP:<RES.STB,OVL4/T:117620,FORLIB[1,1]/L/E

PASS 2
```

The following is the load map of the resident section.

```
LOAD MAP

TRANSFER ADDRESS: 117622
LOW LIMIT: 117622
HIGH LIMIT: 137460
**********
MODULE  MAIN.
SECTION ENTRY    ADDRESS SIZE
<. ABS.>         000000  000000
<      >         117622  000446
         MAIN.   117622
<.$$$$.>         120270  000010
<X     >         120300  000004
**********
MODULE  PRINT
SECTION ENTRY    ADDRESS SIZE
<      >         120304  000170
         PRINT   120304
**********
MODULE  FORCE
SECTION ENTRY    ADDRESS SIZE
<      >         120474  000000
**********
MODULE  $LNK00
SECTION ENTRY    ADDRESS SIZE
<      >         120474  000360
         LINK    120474
         RETURN  120656
**********
MODULE  $RUN00
SECTION ENTRY    ADDRESS SIZE
<      >         121054  001130
         RUN     121054
         STKBAS  122174
**********
MODULE  $LDR00
SECTION ENTRY    ADDRESS SIZE
<      >         122204  000616
         LDR     122204
**********
     .
     .
     .
**********
MODULE  $DCC01
SECTION ENTRY    ADDRESS SIZE
<      >         123072  001652
         $DCC    123104
         $ECC    123134
         $FCC    123100
         $GCC    123072
**********
MODULE  $DCI01
SECTION ENTRY    ADDRESS SIZE
<      >         124744  001400
         $DCI    124752
         $RCI    124744
**********
```

Notes

Resident main program

Blank COMMON
Labelled COMMON

PRINT is a resident subroutine

FORCE is a resident subroutine

LINK subroutine

LINK entry point
RETURN entry point

RUN subroutine

LOAD subroutine.

Other resident subroutines

Note these two forcibly loaded format conversion packages

```
      .
      .
      .
**********
MODULE  $DLCO
SECTION ENTRY   ADDRESS SIZE
<     >         134456  000006
        $LCO    134456
**********
MODULE  $DICO
SECTION ENTRY   ADDRESS SIZE
<     >         134464  000006
        $ICO    134464
        $OCO    134464
**********
MODULE  $DICI
SECTION ENTRY   ADDRESS SIZE
<     >         134472  000006
        $ICI    134472
        $OCI    134472
**********
MODULE  $DLCI
SECTION ENTRY   ADDRESS SIZE
<     >         134500  000006
        $LCI    134500
**********
      .
      .
      .
**********
MODULE  $OTV04
SECTION ENTRY   ADDRESS SIZE
<     >         136210  000032
        $ERRWK  136224
        $EXSW   136222
        $NAMC   136216
        $OTSV   136210
        $SEQC   136220
**********
      .
      .
      .
**********
MODULE  $IBF02
SECTION ENTRY   ADDRESS SIZE
<     >         137212  000246
        $IOBUF  137214
```

Notes

Resident subroutines.

Note the four dummy format conversion packages.

Resident subroutines.

FORTRAN impure area pointer ($OTSV).

Resident subroutines.

FORTRAN Input/Output Buffer.

The following is a load map of the first overlay file.

```
LOAD MAP

TRANSFER ADDRESS: 116656
LOW LIMIT: 116656
HIGH LIMIT: 117620
**********
MODULE  $$$N.
SECTION ENTRY    ADDRESS SIZE
<. ABS.>         000000  000000
<       >        117622  000446
         $$$N.   117622
<.$$$$.>         120270  000010
<X      >        120300  000004
**********
MODULE  PRINT
SECTION ENTRY    ADDRESS SIZE
<       >        120304  000170
         PRINT   120304
**********
MODULE  FORCE
SECTION ENTRY    ADDRESS SIZE
<       >        120474  000000
**********
MODULE  $LNK00
SECTION ENTRY    ADDRESS SIZE
<       >        120474  000360
         LINK    120474
         RETURN  120656
**********
MODULE  $RUN00
SECTION ENTRY    ADDRESS SIZE
<       >        121054  001130
         RUN     121054
         STKBAS  122174
**********
      .
      .
      .
**********
MODULE  $DC001
SECTION ENTRY    ADDRESS SIZE
<       >        123072  001652
         $DCO    123104
         $ECO    123134
         $FCO    123100
         $GCO    123072
**********
MODULE  $DCI01
SECTION ENTRY    ADDRESS SIZE
<       >        124744  001400
         $DCI    124752
         $RCI    124744
**********
```

Notes

Note change of main object module and global name in the resident section of the symbol table.

Resident subroutines.

```
    .
    .
    .
**********
MODULE  $DLCO
SECTION ENTRY   ADDRESS SIZE
<       >       134456  000006
        $LCO    134456
**********
MODULE  $DICO
SECTION ENTRY   ADDRESS SIZE
<       >       134464  000006
        $ICO    134464
        $OCO    134464
**********
MODULE  $DICI
SECTION ENTRY   ADDRESS SIZE
<       >       134472  000006
        $ICI    134472
        $OCI    134472
**********
MODULE  $DLCI
SECTION ENTRY   ADDRESS SIZE
<       >       134500  000006
        $LCI    134500
**********
    .
    .
    .
**********
MODULE  $OTV04
SECTION ENTRY   ADDRESS SIZE
<       >       136210  000032
        $ERRWK  136224
        $FXSW   136222
        $NAMC   136216
        $OTSV   136210
        $SEOC   136220
**********
    .
    .
    .
**********
MODULE  $IBF02
SECTION ENTRY   ADDRESS SIZE
<       >       137212  000246
        $IOBUF  137214
**********
MODULE  MAIN.
SECTION ENTRY   ADDRESS SIZE
<       >       116656  000554
        MAIN.   116656
**********
```

Notes

Resident subroutines.

Resident subroutines.

Resident subroutines.

Overlay 1 main program.

The following is the load map of the second overlay file:

Notes

```
LOAD MAP

TRANSFER ADDRESS: 115154
LOW LIMIT: 115154
HIGH LIMIT: 117620
**********
MODULE  $$$N.
SECTION ENTRY    ADDRESS SIZE
<. ABS.>         000000  000000
<       >        117622  000446
         $$$N.   117622
<.$$$$.>         120270  000010
<X      >        120300  000004
**********
MODULE  PRINT
SECTION ENTRY    ADDRESS SIZE
<       >        120304  000170
         PRINT   120304
**********
MODULE  FORCE
SECTION ENTRY    ADDRESS SIZE
<       >        120474  000000
**********
MODULE  $LNK00
SECTION ENTRY    ADDRESS SIZE
<       >        120474  000360
         LINK    120474
         RETURN  120656
**********
MODULE  $RUN00
SECTION ENTRY    ADDRESS SIZE
<       >        121054  001130
         RUN     121054
         STKBAS  122174
**********
MODULE  $LDR00
SECTION ENTRY    ADDRESS SIZE
<       >        122204  000616
         LDR     122204
**********
     .                                  Resident subroutines.
     .
     .
**********
MODULE  $DC001
SECTION ENTRY    ADDRESS SIZE
<       >        123072  001652
         $DCO    123104
         $ECO    123134
         $FCO    123100
         $GCO    123072
**********
MODULE  $DCI01
SECTION ENTRY    ADDRESS SIZE
<       >        124744  001400
         $DCI    124752
         $RCI    124744
**********
```

```
**********
MODULE  $DLCO
SECTION ENTRY    ADDRESS SIZE
<       >        134456  000006
         $LCO    134456
**********
MODULE  $DICO
SECTION ENTRY    ADDRESS SIZE
<       >        134464  000006
         $ICO    134464
         $OCO    134464
**********
MODULE  $DICI
SECTION ENTRY    ADDRESS SIZE
<       >        134472  000006
         $ICI    134472
         $OCI    134472
**********
MODULE  $DLCI
SECTION ENTRY    ADDRESS SIZE
<       >        134500  000006
         $LCI    134500
**********
     .
     .
     .

**********
MODULE  $OTV04
SECTION ENTRY    ADDRESS SIZE
<       >        136210  000032
         $ERRWK  136224
         $EXSW   136222
         $NAMC   136216
         $OTSV   136210
         $SEQC   136220
**********
     .
     .
     .

**********
MODULE  $IBF02
SECTION ENTRY    ADDRESS SIZE
<       >        137212  000246
         $IOBUF  137214
**********
MODULE  MAIN.
SECTION ENTRY    ADDRESS SIZE
<       >        115154  000164
         MAIN.   115154
**********
     .
     .
     .

**********
MODULE  $SIN02
SECTION ENTRY    ADDRESS SIZE
<       >        115360  000354
         COS     115360
         SIN     115414
**********
     .
     .
     .
```

Notes

Resident subroutines.

Resident subroutines.

Overlay 2 main program.

Overlay 2 subroutines.

Sine routine obtained from FORTRAN Library

Overlay 2 subroutines.

The following is the load map of the third overlay file:

```
LOAD MAP                                                    Notes

TRANSFER ADDRESS: 116040
LOW LIMIT: 116040
HIGH LIMIT: 117620
**********
MODULE  $$SN.
SECTION.ENTRY    ADDRESS SIZE
<. ABS.>         000000  000000
<       >        117622  000446
         $$SN.   117622
<.$$$$.>         120270  000010
<X      >        120300  000004
**********
MODULE  PRINT
SECTION.ENTRY    ADDRESS SIZE
<       >        120304  000170
         PRINT   120304
**********
MODULE  FORCE
SECTION.ENTRY    ADDRESS SIZE
<       >        120474  000000
**********
MODULE  $LNK00
SECTION.ENTRY    ADDRESS SIZE
<       >        120474  000360
         LINK    120474
         RETURN  120656
**********
MODULE  $RUN00
SECTION.ENTRY    ADDRESS SIZE
<       >        121054  001130
         RUN     121054
         STKBAS  122174
**********
MODULE  $LDR00
SECTION.ENTRY    ADDRESS SIZE
<       >        122204  000616
         LDR     122204
**********
    .
    .                                    Resident subroutines.
    .
**********
MODULE  $DCO01
SECTION.ENTRY    ADDRESS SIZE
<       >        123072  001652
         $DCO    123104
         $ECO    123134
         $FCO    123100
         $GCO    123072
**********
MODULE  $DCI01
SECTION.ENTRY    ADDRESS SIZE
<       >        124744  001400
         $DCI    124752
         $RCI    124744
**********
```

```
**********
MODULE  $DLCO
SECTION ENTRY    ADDRESS SIZE
<       >        134456  000006
         $LCO    134456
**********
MODULE  $DICO
SECTION ENTRY    ADDRESS SIZE
<       >        134464  000006
         $ICO    134464
         $OCO    134464
**********
MODULE  $DICI
SECTION ENTRY    ADDRESS SIZE
<       >        134472  000006
         $ICI    134472
         $OCI    134472
**********
MODULE  $DLCI
SECTION ENTRY    ADDRESS SIZE
<       >        134500  000006
         $LCI    134500
**********
     .
     .
     .

**********
MODULE  $OTV04
SECTION ENTRY    ADDRESS SIZE
<       >        136210  000032
         $ERRWK  136224
         $EXSW   136222
         $NAMC   136216
         $OTSV   136210
         $SEQC   136220
**********
     .
     .
     .
**********
MODULE  $IBF02
SECTION ENTRY    ADDRESS SIZE
<       >        137212  000246
         $IOBUF  137214
**********
MODULE  MAIN.
SECTION ENTRY    ADDRESS SIZE
<       >        116040  000202
         MAIN.   116040
**********
     .
     .
     .
**********
MODULE  $SQT02
SECTION ENTRY    ADDRESS SIZE
<       >        116262  000136
         SQRT    116262
**********
     .
     .
     .
```

Notes

Resident subroutines.

Resident subroutines.

Overlay 3 main program.

Overlay 3 subroutines.

Square root routine obtained from FORTRAN Library.

Overlay 3 subroutines.

The following is the load map of the fourth overlay file:

```
                                                        Notes

LOAD MAP

TRANSFER ADDRESS: 117300
LOW LIMIT: 117300
HIGH LIMIT: 117620
**********
MODULE  $$$N.
SECTION ENTRY   ADDRESS SIZE
<. ABS.>        000000  000000
<       >       117622  000446
        $$$N.   117622
<.$$$$.>        120270  000010
<X      >       120300  000004
**********
MODULE  PRINT
SECTION ENTRY   ADDRESS SIZE
<       >       120304  000170
        PRINT   120304
**********
MODULE  FORCE
SECTION ENTRY   ADDRESS SIZE
<       >       120474  000000
**********
MODULE  $LNK00
SECTION ENTRY   ADDRESS SIZE
<       >       120474  000360
        LINK    120474
        RETURN  120656
**********
MODULE  $RUN00
SECTION ENTRY   ADDRESS SIZE
<       >       121054  001130
        RUN     121054
        STKBAS  122174
**********
MODULE  $LDR00
SECTION ENTRY   ADDRESS SIZE
<       >       122204  000616
        LDR     122204
**********
    .
    .
    .                                   Resident subroutines.
**********
MODULE  $DC001
SECTION ENTRY   ADDRESS SIZE
<       >       123072  001652
        $DCO    123104
        $ECO    123134
        $FCO    123100
        $GCO    123072
**********
MODULE  $DCI01
SECTION ENTRY   ADDRESS SIZE
<       >       124744  001400
```

```
          $DCI     124752
          $RCI     124744
**********
     .
     .
     .
**********
MODULE  $DLCO
SECTION ENTRY    ADDRESS SIZE
<       >        134456  000006
          $LCO   134456
**********
MODULE  $DICO
SECTION ENTRY    ADDRESS SIZE
<       >        134464  000006
          $ICO   134464
          $OCO   134464
**********
MODULE  $DICI
SECTION ENTRY    ADDRESS SIZE
<       >        134472  000006
          $ICI   134472
          $OCI   134472
**********
MODULE  $DLCI
SECTION ENTRY    ADDRESS SIZE
<       >        134500  000006
          $LCI   134500
**********
     .
     .
     .
**********
MODULE  $OTV04
SECTION ENTRY    ADDRESS SIZE
<       >        136210  000032
          $ERRWK 136224
          $EXSW  136222
          $NAMC  136216
          $OTSV  136210
          $SEQC  136220
**********
     .
     .
     .

**********
MODULE  $IBF02
SECTION ENTRY    ADDRESS SIZE
<       >        137212  000246
          $IOBUF 137214
**********
MODULE  MAIN.
SECTION ENTRY    ADDRESS SIZE
<       >        117300  000320
          MAIN.  117300
```

Resident subroutines.

Resident subroutines.

Resident subroutines.

Overlay 4 main program.

The following disk directory shows the results of linking the overlay system:

```
$RU PIP

PIP-11  V005B
#/DI

DIRECTORY DF0: [200,200]
08-DEC-71

DUMMY .OBJ      2  08-DEC-71 <233>
PRINT .OBJ      7  08-DEC-71 <233>
RES   .OBJ     14  08-DEC-71 <233>
OVL1  .OBJ     16  08-DEC-71 <233>
OVL2  .OBJ      8  08-DEC-71 <233>
OVL3  .OBJ      8  08-DEC-71 <233>
OVL4  .OBJ     11  08-DEC-71 <233>
RES   .STB     14  08-DEC-71 <233>
RES   .LDA     93  08-DEC-71 <233>
OVL1  .LDA      8  08-DEC-71 <233>
OVL2  .LDA     17  08-DEC-71 <233>
OVL3  .LDA     12  08-DEC-71 <233>
OVL4  .LDA      4  08-DEC-71 <233>

TOTL BLKS:   214
TOTL FILES:   13
```

Shown below is the run-time output of this overlay system. The output clearly traces the control path through the system.

```
$RU RES
RESIDENT
OVERLAY1
INPUT
+1.00
OK

LABEL2
OVERLAY2
LABEL3
```

```
OVERLAY3
OVERLAY4
A= 1.00       SIN(A)= 0.84       SQRT SIN(A)= 0.92
LABEL4
RESIDENT
OVERLAY1
INPUT
+1.57
OK

LABEL2
OVERLAY2
LABEL3
OVERLAY3
OVERLAY4
A= 1.57       SIN(A)= 1.00       SQRT SIN(A)= 1.00
LABEL4
RESIDENT
OVERLAY1
INPUT
↑C
.KI
$
```

CHAPTER 7

LINK-11 UNDER RSX-11C

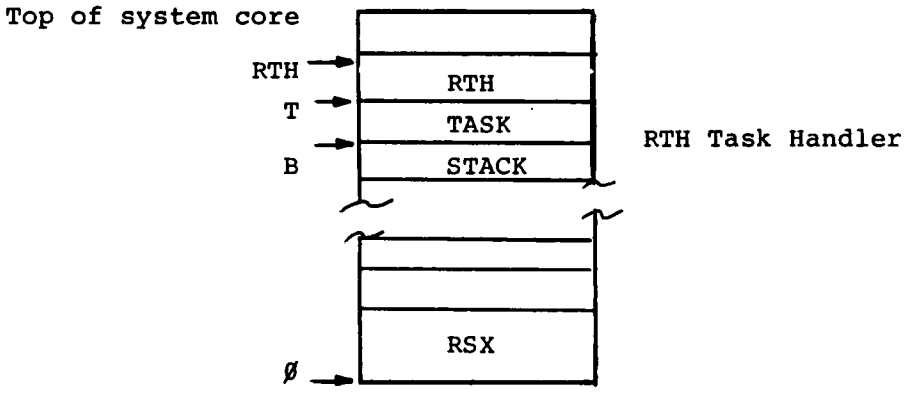
The Linker supplied with RSX is the non-segmented version with no monitor library search function. The Linker searches libraries on paper tape and has a real-time header (RTH) generation capability.

7.1 TASK LIMITS

Under DOS, the initially-defined top of core is the top of physical memory less the space for the bootstrap and absolute loaders ($32\emptyset_8$ bytes). Under RSX, when a task asks the Executive for the address of the top of memory, the top of the requesting task is returned, less the space used by the RTH.

Therefore, the initially-defined top for the Linker under RSX is the top of the Linker minus $32\emptyset_8$ bytes. If no /T or /B switch appears in the Linker command string, the new task being linked is linked at the currently-defined top of core, i.e., the top of the Linker as described above. This means that the /T or /B switch should always be used to force a specific address during the linking procedure.

In the diagram of an RSX core memory, the RSX Executive is shown stored at the bottom of core. A task requesting its top address is given address T, although its RTH starts at address RTH. The bottom address is actually the address of the bottom of the task stack and is returned as address B.



7.2 DEVICE NAMES

The Linker uses the following logical names and default physical names:

Logical Device Name	Physical Device Name	Use
CMI	KB	Command String Input
CMO	KB	Linker generated error message (not IOT generated) and for the "#" printed to solicit command string input
MAP	-	Load map output
OBJ	-	Object module input
LOD	-	Load module output
STB	-	Core Library directory output

However, the Linker will not operate properly if the ASSIGN monitor command is used.

7.3 RELINKING THE LINKER

Link-11 is supplied as a task as part of the basic RSX-11 kit and as an object module for re-linking. It should be re-linked with the following command string:

```
#PP:,LP:<PR:/T:XXXX
#/RT:LINK:::65535:YYYY/E
```

where "XXXX" is the desired top and "YYYY" specifies a stack size large enough for the Linker's stack, command strings, and symbol table. The Linker requires a stack of 1ØØ bytes plus 1Ø bytes per entry in the symbol table plus 1ØØ bytes per command string. If a load map is not desired, leave ",LP:" out of the first command line. If desired, a bottom address (/B switch) can be specified rather than a top.

7.4 REAL-TIME HEADER GENERATION

This Section covers the processes for generating real-time headers (RTH). The contents of the RTH is discussed in Chapter 1Ø of the RSX-11 User's Guide.

A task with a RTH and stack is generated by Link-11 in a manner similar to linking any program except that a special command string switch (/RT) is used. The form of the switch is:

```
/RT:NAME:CN:PR:MAX:STK
```

where:

NAME	is the symbolic name of the task.
CN	is the call number of the task.
PR	is the task priority level in the range 1-4.
MAX	is the maximum level 1 run time, in seconds, before an error message is generated. A Ø value means there is no time-out. A negative value specifies the task as a background task.
STK	is the stack size in bytes. This value is increased by 2 to allow for the stack overflow check word.

When the /RT switch is used, the Linker supplies default values for all values not supplied, as follows:

NAME	the load module name which is the first object module name seen. For FORTRAN, the default is "MAIN.".
CN	the low limit of the task. This is an arbitrary unique number.
PR	level 1.
MAX	Ø (no limit).
STK	70_{10} words (140_{10} bytes). This is the minimum size required by a FORTRAN task with no subroutines.

Any trailing set of values can be omitted and default values selected. Individual default values can be selected by using two successive colons. For example:

```
#/RT:ABC::3
```

selects default values for the call number, maximum time and stack size.

The /RT switch, if used, must appear as the last command string line to the Linker and must be followed by /E. This switch can be used in two ways. The form:

```
#/RT.../E
```

instructs the Linker to create a RTH with no inputs other than the switch values (if any) and default values. The form:

```
#DEV:/RT.../E
```

indicates that an RTH is being supplied in object module form. As discussed in the RSX-11 User's Guide, the object module may include values for entries in the RTH. These values can be overridden by supplying new values with the /RT switch. Note that when the RTH is being supplied as an object module, the /RT switch must be used even if no values are provided with the switch. The /RT informs the Linker that the object module is a RTH and that special actions are necessary. Thus, for example, a RTH in object module form is processed (having no changes) with the following command:

```
#PR:/RT/E
```

7.5 CORE LIBRARY SYMBOL TABLE

Core Libraries are described in the RSX User's Guide (see Section 10.5). This section reviews the processes of generating and reading the Core Library directories with the Linker.

7.5.1 Core Library Linker Procedures

A Core Library directory is generated by the Linker as an optional third output file. Thus, the command:

```
#PP:,,PP:<PR:/B:XXXX/E
```

punches the Core Library directory followed by the Core Library load module. These two files are separated by leader/trailer and can easily be broken into two tapes.

On subsequent linking, the Linker reads the directories with no special switches required; the Core Library directories are read first followed by the object modules of the task to be linked to the Core Library. Thus, assuming that a task consists of a single object module to be linked to a single Core Library, the command string would be:

```
#PP:<PR:,PR:
```

or

```
#PP:<PR:/TA:2
```

The core library directories are read only on pass 1 of the Linker and must not be put in the reader on pass 2.

PART II

LIBR-11 LIBRARIAN

Chapter 8.	Introduction	8-1
Chapter 9.	Operating Procedures	9-1
9.1	Calling Libr-11	9-1
9.2	Command Strings	9-1
9.2.1	Creating a Library	9-1
9.2.2	Updating a Library	9-2
9.2.2.1	To Delete One or More Object Modules	9-2
9.2.2.2	To Insert One or More Object Modules	9-3
9.2.2.3	To Replace One or More Object Modules	9-3
9.2.3	Listing a Library	9-3
9.2.4	Naming Libraries	9-5
9.2.5	Legal File Specification Combinations	9-5
9.3	Examples	9-6
9.4	Error Messages	9-8
9.5	Programming Notes	9-9
9.5.1	Creating	9-9
9.5.2	Updating	9-9
9.5.3	Listing	9-9
Appendix A	Sample Links	A-1

CHAPTER 8

INTRODUCTION TO LIBR-11 LIBRARIAN

The PDP-11 Librarian (Libr-11) is a system program for the Disk Operating System. Libr-11 provides facilities for creating, modifying, deleting, and listing the contents of libraries. A library is a file which consists of one or more object modules. (An object module is the binary output of the DOS Assembler.)

Libr-11 is a valuable program for the DOS user because:

- It eliminates having separate directory entries in a User File Directory (UFD) for each object module.
- It expedites the linking process in conjunction with the Linker's library search capabilities.
- It allows for standardization and controlled updating of frequently used routines, e.g., FORTRAN cosine routine.

The user controls the operation of Libr-11 through command strings typed on the keyboard. Specified in the command strings are such things as devices, library and object module names, and switches which indicate the Libr-11 operation desired. The user can direct Libr-11 to:

- Create a library
- Update a library
- Insert one or more object modules in a library
- Replace one or more object modules in a library
- List the contents of a library
- Delete one or more object modules from a library
- Delete an entire library

A directory listing of the object modules of a library can be obtained merely by specifying the device on which the directory is to appear and the name of the library.

The flexibility of Libr-11 enables the user to specify certain combinations of operations in a single command string. For example,

a library can be modified, renamed, and listed in one command string.

The switch options which direct Libr-11's operations are:

Switch	Operation
/D	Delete object module
/DL	Delete input library
/I	Insert object module
/LO	List object modules
/R	Replace object module

If you type an illegal command string, e.g., illegal format, excessive switches, nonexistent file or object module, etc., Libr-11 will print an appropriate error message on the teleprinter.

The following discussion assumes that the reader is familiar with the DOS Monitor, Edit-11 Text Editor, DOS Assembler, ODT-11R Debugging Program, and Link-11 Linker.

CHAPTER 9

OPERATING PROCEDURES, LIBR-11

9.1 CALLING LIBR-11

The Librarian is called into core by typing the RUN command in response to the DOS Monitor's dot or dollar sign. (The Librarian's call name can be determined by listing the system directory using PIP.) The Librarian is often stored as LIBR, and when called it prints its name, version number, and a # sign, and then waits for the user to issue a command string. For example:

```
.RUN LIBR      <CR>
LIBR-11    VØØ1A
#
```

9.2 COMMAND STRINGS

When the Librarian is in core and has printed the # sign, it is ready to accept a user command string. The format is:

output library, listing file < input library, input file(s)

Libr-11 performs two passes over all input files. For nonfile-structured devices (e.g., paper tape reader), the system will inform the user to reload the device for the second pass. For file-structured devices, both passes are performed automatically without requiring any user intervention.

9.2.1 Creating a Library

output library (,listing file) < ,input file(s)

A library is created on the device specified in the output library specification and named as specified. The listing file specification is optional and, if present, the contents of the output library will be listed. The format of the listing will be fully discussed later.

An input library need not appear, but the comma and one or more input files must appear (each of which contains one or more object modules*). For example:

```
#DT1:FIL.LIB<,FIL.1,FIL.2      <CR>
```

*Note that an input file of concatenated object modules differs from a library in that it does not have a directory of the object modules that it contains.

would create a library named FIL.LIB on DECtape 1. The library would consist of all object modules in FIL.1 and FIL.2 in that order, and in the order in which the object modules appear in their respective input files.

A restriction placed on the user by the Librarian is that it accepts a command specification of only one line. It is therefore suggested when creating a Library from a large number of object modules, that the object modules be concatenated into a reasonable number of files. An example of this is as follows.

```
$RU PIP
#FILE.ØØ1<FILE.A,FILE.B,FILE.C)
#FILE.ØØ2<FILE.D,FILE.E,FILE.F)
#FILE.ØØ3<FILE.G,FILE.H,FILE.I)
#↑C
.KI
$RU LIBR
#NAME.LIB<,FILE.ØØ1,FILE.ØØ2,FILE.ØØ3)
```

9.2.2 Updating a Library

Libraries can be updated in one of three ways:

9.2.2.1 To Delete One or More Object Modules

output library (,listing file) < input library/D:v_1:...v_n

The output library will be created as a result of deleting the object modules named v_1...,v_n from the input library. The listing file is optional.

The name associated with an object module is the symbol assigned to the module by the DOS Assembler's .TITLE directive.

The object modules to be deleted must appear in the same order as they appear in the library; their order can be determined from the listing.

Insert and/or Replace operations cannot accompany a Delete request. For example:

```
#DT1:LIBR.1 < DT2:LIBR.Ø/D:M1:M2     <CR>
```

would create a library named LIBR.1 on DECtape 1 as a result of deleting the object modules M1 and M2 from LIBR.Ø on DECtape 2.

9.2.2.2 To Insert One or More Object Modules

```
output library (,listing file)<input library,input file(s)/I(:v)
```

The output library will be created as a result of inserting the object modules of the input file into the input library. If v is specified, the object module(s) in the input file are inserted starting at position v, otherwise, they are inserted at the end. v is treated as a decimal integer, and is always relative to the input library.

If more than one input file is specified for insertion, the positions at which they are to be inserted must appear in non-descending order. For example:

```
#DT1:LIBR.1<DT2:LIBR.Ø,FIL.1/I:2,FIL.2/I      <CR>
```

would create an output library on DECtape 1 as a result of inserting the object modules of FIL.1 into LIBR.Ø, beginning at position 2, and then inserting the object modules of FIL.2 into LIBR.Ø at the end.

Insert and Replace operations can appear in the same command so long as the order restriction is observed.

9.2.2.3 To Replace One or More Object Modules

```
output library (,listing file)<input library,input file(s)/R
```

The output library will be created as a result of replacing the object module(s) in the input library by those in the input file(s).

The object modules to be replaced must have the same name as those which replace them, and they must correspond orderwise. For example:

```
#DT1:LIBR.1<DT2:LIBR.Ø,FIL.1/R,FIL.2/R      <CR>
```

would create the output library LIBR.1 on DECtape 1 as a result of replacing the object modules in the input library LIBR.Ø with those in FIL.1 and FIL.2.

9.2.3 Listing a Library

```
,listing file(/LO)<input library
```

The directory of the input library will be listed. Optionally, the presence of the /LO switch directs the Librarian to produce an object module listing. This is intended as a means to double-check the accuracy of the library; the directory listing must correspond exactly to the object module listing.

The output library will be listed when one was created; otherwise, the input library will be listed. The format of the listing is:

```
Library Name & Extension

Decimal Order Number        Object Module Name (1st Module)
        :                           :
Decimal Order Number        Object Module Name (last module)
```

For example, if LIB.1 contains object modules M1, M2, and M3 in that order, the command:

```
#,LP:FIL.LST<DT1:LIB.1      <CR>
```

will produce on the line printer:

```
FIL     .LST

00001     M1
00002     M2
00003     M3
```

If the /LO switch appears, for example:

```
#,LP:FIL.LST/LO<DT1:LIB.1      <CR>
```

the listing above would be followed by a form feed and a similar table, except that the name of the second table is always OBJMOD.LST.

The library name that is printed at the head of the listing is the name specified in the listing file specification. For example:

```
#LIB.ABC,LP:NAME<,FIL.1,FIL.2      <CR>
```

The listing would be titled NAME, not the new created library LIB.ABC. When the listing file name is not specified, then the listing would be titled with the name of the newly created file.

9.2.4 Naming Libraries

The output library can have the same name as the input library. In this case, however, the input library has an implied /DL; that is, the input library is deleted. For example:

```
#LIB.1<LIB.1/D:OM1      <CR>
```

is the same as:

```
#LIB.TMP<LIB.1/D:OM1/DL      <CR>
```

and then rename LIB.TMP to LIB.1.

CAUTION

The user should never name a Library LIBR.TMP. This name is reserved for use by the Librarian.

9.2.5 Legal File Specification Combinations

In a command string, various combinations of file specifications are possible; legal combinations and their operation are shown below.

	Output Library	Listing	Input Library	Input File(s)	Operation	Note
(1)	P	P	P	P	Insert or Replace Object Modules; List Output Library	SE if /D on input library
(2)	P	P	P	NP	Delete Object Modules; List Output Library	SE if /D not on input library
(3)	P	P	NP	P	Create Library; List Output Library	SE if switch on input file
(4)	NP	P	P	NP	List Input Library	SE if /D on input library
(5)	P	NP	P	P	Same as (1) except no listing	Same as (1)
(6)	P	NP	P	NP	Same as (2) except no listing	Same as (2)
(7)	P	NP	NP	P	Same as (3) except no listing	Same as (3)

Legend: P = present
NP = not present
SE = syntax error

9.3 EXAMPLES

Assume FIL.1 contains object modules OM1, OM2, and OM3 in that order, FIL.2 contains OM4 and OM5 in that order, FIL.3 contains OM5 and OM3 in that order, and FIL.4 contains OM6. Then:

```
#LIB.1,LP:LIB.1<,FIL.1,FIL.2
```

will create a library named LIB.1 containing object modules OM1, OM2, OM3, OM4, and OM5 in that order. The listing will appear on the line printer as:

```
LIB     .1

ØØØØ1     OM1
ØØØØ2     OM2
ØØØØ3     OM3
ØØØØ4     OM4
ØØØØ5     OM5
```

Files FIL.1 and FIL.2 remain unaltered. The listing is produced after all other actions have been performed. Consequently,

```
#LIB.1,LIB.1<,FIL.1,FIL.2      <CR>
```

would produce an error message (file already exists) when an attempt is made to write the listing to the disk.

Using the assumption above:

```
#LIB.2<LIB.1/D:OM1:OM4      <CR>
```

will create a library named LIB.2 containing object modules OM2, OM3, and OM5 in that order. No listing is produced and LIB.1 is not deleted.

```
#LIB.3<LIB.2/D:OM3:OM2      <CR>
```

will produce an error message because the modules to be deleted are not in the order in which they appear in the library.

The command string:

```
#,LP:LIB2.LS/LO<LIB.2      <CR>
```

will produce a listing on the line printer which appears as:

```
LIB2    .LS

ØØØØ1      OM2
ØØØØ2      OM3
ØØØØ3      OM5
<form feed>
OBJMOD.LST

ØØØØ1      OM2
ØØØØ2      OM3
ØØØØ3      OM5
```

The command string:

```
#LIB.3<LIB.2/DL,FIL.4/I:2      <CR>
```

will create a library named LIB.3 containing OM2, OM6, OM3 and OM5 in that order. No listing is produced and LIB.2 is deleted.

The command string:

```
#LIB.4<LIB.3,FIL.4/R      <CR>
```

will create a library named LIB.4, which is really LIB.3 with OM6 replaced (i.e., removed from LIB.3 before creating LIB.4).

```
#LIB.5< LIB.4,FIL.3/R      <CR>
```

will produce an error message because the object modules in FIL.3 are not in the same order as in LIB.4.

The command string:

```
#LIB.5<LIB.3/DL,FIL.4/I      <CR>
```

will create a library named LIB.5 containing OM2, OM6, OM3, OM5 and OM6 in that order. No listing is produced and LIB.3 is deleted. Note that a library can contain multiple copies of the same object module, e.g., two OM6 modules, above.

The command string:

```
#LIB.6<LIB.5/D:OM6      <CR>
```

will create a library named LIB.6 containing OM2, OM3, OM5 and OM6 in that order. No listing is produced and LIB.5 is not deleted. When a library contains multiple copies of the same object module, they are deleted one at a time in their order of occurrence.

If the purpose of the previous example were to delete all occurrences of OM6, the command string would have been either:

```
#LIB.6<LIB.5/D:OM6:OM6      <CR>
```

or

```
#LIB.6<LIB.5/D:OM6/D:OM6     <CR>
```

9.4 ERROR MESSAGES

Error messages issued by the Librarian are listed below. (See the DOS Monitor document for a listing of all DOS error messages.)

Error Code	Additional Information	Meaning
S2Ø2	File Name and Error Status Byte	Fatal I/O error; due to truncated line, checksum, character parity, or device parity error.
S2Ø3	File Name	Switch error or semantic error; due to illegal switch, too many switches on a file, or illegal combination of file specifications.
S2Ø4		Illegal file specification format; more than two output files specified.
S213	File Name	Error on input file; illegal object module format; first line not a GSD, or EOF prior to reading end module line.
S244	File Name	Out of order; already past requested position for Insert.
S245	File Name	Object module error; object module not found, or /R or /D out of order.
S246		Error on input library; illegal library format, first two lines incorrect.
S247		Listing error; output library cannot be read from output library device, i.e., PP:.

9.5 PROGRAMMING NOTES

9.5.1 Creating

Creating a library requires an output library specification and the input files to be placed in the library. The input files must be preceded by a comma; so they are not considered to be an input library. Listing the newly created library is optional.

Because the length of the Librarian's command string is restricted to one (1) Teletype line, creating libraries from many object modules must be accomplished with an intermediate step.

The step is to first concatenate the object modules in their desired order using PIP. Having done this, the command string to the Librarian is reduced to one input file, i.e., that containing the concatenated object modules.

9.5.2 Updating

To update a library, both output and input library specifications are required, along with object modules or input files. Updating includes deletion of object modules, insertion of object modules, and replacing of one or more object modules. Listing the updated library is optional.

9.5.3 Listing

To list the contents of a library only requires a listing file specification and an input library specification. The listing file specification must be preceded by a comma, so that it is not interpreted as an output library.

APPENDIX A

SAMPLE LINKS

1. FORTRAN

User is logged in under user identification code (UIC) of 200,200.

The system device is on RF11 disk. He wishes to link a FORTRAN program (FORT1.OBJ) to the FORTRAN library (FORLIB.OBJ) which is on the system disk under UIC 1,1. He wants a load map printed on the line printer. Input comes from and output goes to the disk.

The command string is:

```
#DF:,LP:<FORT1,FORLIB[1,1]/L/E
```

(Note the default output file name is the first input file name. The default input extension is OBJ. since on the system disk both files FORT1, and FORLIB had the extension OBJ there was no need to put this information in the command strings. The Linker assumes them.)

2. ASSEMBLY LANGUAGE

User is logged in under UIC 200,200. He has a DECtape, but he has no line printer at his installation. He wants his outputs, load module (LOAD.LDA) and load map (LOAD.MAP) on DECtape and his inputs to come from disk. (He has seven input files all with extension OBJ.)

The command strings are:

```
#DT1:LOAD,LOAD<IN1,IN2,IN3
#IN4,IN5,IN6,IN7/E
```

(Note the Linker accepts multiple command strings.) On the DECtape the load module has the extension LDA and the load map MAP.

3. OVERLAYS

The user is logged in under UIC 200,200. The system device is an RF11 disk. He has a line printer. He has three input files; one called RES.OBJ, which is the input file of the resident section; the other input files are OVL1.OBJ, and OVL2.OBJ, which are the input files of the overlays.

The command string to generate the load module, the symbol table, and the load map of the resident section is as follows:

```
#RES,LP:<RES/OV:2,FORLIB[1,1]/L/E
 (DF:RES.LDA)  (DF:RES.OBJ)  (DF:FORLIB.OBJ)
```

NOTE

The full file specifications, including the assumed extensions and devices, are shown in parentheses. The /OV:2 switch specifies two associated overlays.

The command strings to generate the load modules of the overlay files are:

```
#OVL1,LP:<OVL1,FORLIB[1,1]/L/E

#OVL2,LP:<OVL2,FORLIB[1,1]/L/E
```

The /OV:2 value means that the link in which it (/OV:2) appears is the resident link, while the next two links are the associated overlays. At resident link time, the Linker automatically outputs the resident symbol table and puts the program low-limit of the resident load module into this symbol table.

When linking the first overlay, the Linker reads the resident symbol table into core. This table permits the overlay file to reference globals which exist in the resident link. It also contains the resident module's low-limit, which establishes the top of the overlays and uses it until the entire overlay system is linked. In this example, the overlay system consists of one resident and two overlays.

When the count supplies in the /OV switch is exhausted, the Linker reverts to normal (non-overlay) operation.

Absolute program sections, 1-2
Absolute loader, 2-1
Action request messages, Linker, 4-3
Address assignment, 1-2
Advantages, Link-11, 1-1
Angle brackets (< >), writing convention, 3-1
Assembler directive
  .ASECT, 1-2
  .CSECT, 1-2
  .TITLE, 9-2
Assembly language overlays, 6-11
ASSIGN command, 6-14

BEGIN command, 6-13
Bottom (B) switch, 1-3, 3-2

CALL LINE statement, 6-6, 6-7
CALL RETURN statement, 6-6, 6-7
Calling
  Libr-11, 9-1
  overlay file, 6-5
Colon (:) usage, 7-4
Command string
  Libr-11, 9-1
  Linker, 3-1, 6-8
COMMON, 6-2
  communication, 6-12
  reinitializing, 6-14
Communications directory (COMD), 2-1
Concatenate (CC) switch, 3-4
Control section, named or unnamed, 1-2
Core library, 2-3, 7-4
<CR>symbol, writing convention, 8-2
Creating
  core resident module, 6-8
  library, 9-1, 9-4, 9-9
  overlay files, 6-8
.CSECT assembler directive, 1-2
CTRL/C BEGIN, 4-1
CTRL/C RESTART, 4-1

Deleting object modules, 9-2
Device names, 7-2
Directory file, core library, 2-3
DOS assembler, 1-1
DOS Monitor loader, 2-2

.END statement, 2-2
Entry symbols, 1-3
Error handling, Link-11, 4-1
  messages, 4-1, 4-2, 4-3
  syntactical errors, 3-2
Error messages
  Libr-11, 8-2, 9-6, 9-8
  overlay facility, 6-9, 6-10
Examples of file specification combinations, 9-5
Exit (E) switch, 3-5
Exit from overlay, 6-12
External symbols, 1-3

Fatal linking error messages, 4-3
File specification combinations, Libr-11, 9-5
  examples, 9-6
File specifications, Link-11, 3-1
Format conversion, 6-2, 6-3
  table, 6-4
Format, Libr-11 command string, 9-1
FORTRAN system, 6-1

Global declarations, 6-11
Global requests in overlay files, 6-3
Global symbol directory (GSD), 2-1
Global symbols, 1-3

Header generation, real-time, 7-3

Input library, 9-4
Input module, 2-1
Inserting object modules, 9-3
Internal symbols, 1-3
Introduction to Libr-11 Librarian, 8-1

Legal file specification combinations, Libr-11, 9-5
Libr-11 Librarian introduction, 8-1
Library (L) switch, 3-5, 3-6
Library
  creation, 9-1
  file specifications, 9-5
  listing, 9-3, 9-9
  Monitor, 3-6
  naming, 9-5
  searches, 3-6
  update, 9-2
  user, 3-6
LIBR.TMP, 9-5
Link-11 advantages, 1-1

LINK as run-time overlay supervisor, 6-1, 6-4
Linker, RSX, 7-1
Listing library contents, 9-6, 9-9
Loader
  absolute, 2-1
  DOS Monitor, 2-2
Loading, Link-11, 3-1
Load map, Link-11, 2-2
Load module, 2-1

Modules
  input, 2-1
  load, 2-1
  object, 2-1
  output, 2-1
Monitor library, 3-6

Named control section, 1-2
Naming libraries, 9-5
Number sign (#) usage by Link-11, 3-1

Object module, 2-1, 8-1
  deleting, 9-2
  inserting, 9-3
  replacing, 9-3
OBJMOD.LST, 9-4
ODT switch, 3-4
Operating procedures
  Libr-11, 9-1
  Linker, 3-1
Overlay
  communication, 6-2
  exit, 6-12
  facility features, summary, 6-14
  files, 6-5 to 6-8
  register usage, 6-12
  system creation, programming example, 6-15
  transfer paths, 6-6, 6-7
Overlays, assembly language, 6-11
Output library, 9-3
Output module, 2-1

Pass 1, 2-1, 3-4
Pass 2, 2-1, 3-4
PIP, 9-1
Program sections, absolute and relocatable, 1-2
Program overlay facility, 6-1
Programming example, overlay system creation, 6-15
Programming notes
  overlay, 6-13
  switches, 3-7
Protected files, 6-13

Quotes usage, single ('), 6-5

Real-time header generation, 7-3
Register usage, overlay, 6-12
Relinking
  Link-11, 1-3
  RSX Linker, 7-2
Relocatable program sections, 1-2
Replacing object modules, 9-3
Restarting Link-11, 4-1
RUN command, 9-1
Run-time supervisor, 6-1, 6-7

Sharing data and/or programs among object modules, 1-2
Single quote (') usage, 6-5
Switches,
  Bottom (B), 1-3, 3-2
  Concatenate (CC), 3-4
  Exit (E), 3-5
  general notes, 3-5
  Library (L), 3-5, 3-6
  ODT, 3-4
  options, Libr-11, 8-2
  programming notes, 3-7
  summary, Link-11, 5-1
  Tape (TA), 3-3
  Top (T), 1-3, 3-2
  TRansfer address (TR), 3-4
  Undefined globals (U), 3-3
Stack
  overflow, 6-7
  usage, 6-11
Symbols
  entry, 1-3
  external, 1-3
  global, 1-3
  internal, 1-3
Syntactical error, 3-2

Tape (TA) switch, 3-3
.TITLE assembler directive, 9-2
Top address, Link-11, 1-3
Top of core for Linker under RSX, 7-1
Top (T) switch, 1-3, 3-2
Transfer address (TR) switch, 3-4

Undefined globals (U) switch, 3-3
Underlining, writing convention, 3-1
Unnamed control section, 1-2
Updating a library, 9-2, 9-9
User libraries, 3-6

Warning error messages, Link-11, 4-1
Writing conventions, 3-1

HOW TO OBTAIN SOFTWARE INFORMATION

Announcements for new and revised software, as well as programming notes, software problems, and documentation corrections are published by Software Information Service in the following newsletters.

Digital Software News for the PDP-8 & PDP-12
Digital Software News for the PDP-11
Digital Software News for the PDP-9/15 Family

These newsletters contain information applicable to software available from Digital's Program Library, Articles in Digital Software News update the cumulative Software Performance Summary which is contained in each basic kit of system software for new computers. To assure that the monthly Digital Software News is sent to the appropriate software contact at your installation, please check with the Software Specialist or Sales Engineer at your nearest Digital office.

Questions or problems concerning Digital's Software should be reported to the Software Specialist. In cases where no Software Specialist is available, please send a Software Performance Report form with details of the problem to:

Software Information Service
Digital Equipment Corporation
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754

These forms which are provided in the software kit should be fully filled out and accompanied by teletype output as well as listings or tapes of the user program to facilitate a complete investigation. An answer will be sent to the individual and appropriate topics of general interest will be printed in the newsletter.

Orders for new and revised software and manuals, additional Software Performance Report forms, and software price lists should be directed to the nearest Digital Field office or representative. U.S.A. customers may order directly from the Program Library in Maynard. When ordering, include the code number and a brief description of the software requested.

Digital Equipment Computer Users Society (DECUS) maintains a user library and publishes a catalog of programs as well as the DECUSCOPE magazine for its members and non-members who request it. For further information please write to:

DECUS
Digital Equipment Corporation
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754

8.8C